DTIC FILE COPY

(2)

UNCL SECURITY ER COPY - FOR REPRODUCTION PURPOSES

DOCUMENTATION PAGE

EPO AD-A223 174

1b. RESTRICTIVE MARKINGS	
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT: Approved for public release; distribution unlimited.
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S): ARO 24941.16-MA-SDI
6a. NAME OF PERFORMING ORGANIZATION: SYRACUSE UNIVERSITY; 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION: U. S. Army Research Office
6c. ADDRESS (City, State, and ZIP Code): SYRACUSE, NEW YORK 13244-5300	7b. ADDRESS (City, State, and ZIP Code): P. O. Box 12211, Research Triangle Park, NC 27709-2211
8a. NAME OF FUNDING / SPONSORING ORGANIZATION: U. S. Army Research Office; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: DAAL03-87-K-0027
8c. ADDRESS (City, State, and ZIP Code): P. O. Box 12211, Research Triangle Park, NC 27709-2211	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO

ELECTE JUN 15 1990 S D

11. TITLE (Include Security Classification)

TOEPLITZ MATRICES: ALGEBRA AND ALGORITHMS (UNCLASSIFIED)

12. PERSONAL AUTHOR(S)
HARI KRISHNA

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
FINAL	FROM 2-87 TO 1-90	MARCH 31, 1990	137

16. SUPPLEMENTARY NOTATION
The view, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Toeplitz matrices, computational complexity, fast and superfast algorithms, numerical stability, finite precision analysis, wide sense stability of systems, predictor polynomial

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

In this research project, we analyze the mathematical structure and numerical algorithms associated with Toeplitz matrices. Toeplitz matrices arise in a number of problems in engineering and applied mathematics. In many such problems, the task is to solve for certain parameters of interest (such as predictor polynomial, reflection coeffients, and solution to a linear system) in a computationally effcient manner. Also, the numerical stability aspects of the various algorithms must be examined from the standpoint of implementation using finite precision arithmetic. We have derived fast (order-recursive) and superfast (fast Fourier transform based) algorithms for solving a Toeplitz linear system. The algorithms reported here are some of the most computationally efficient algorithms. Also, the numerical stability of the split Levinson algorithm is examined and it is established that it is weakly stable. Furthermore, the various classical and split Levinson algorithms are studied for the effects of finite precision arithmetic. An interesting relationship between Levinson algorithm and stability tests for discrete systems is exploited to derive a new computationally efficient algorithm for testing the wide sense of stability of discrete time systems

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code)	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

ITEM 18: reflection coefficients, linear systems.

TOEPLITZ MATRICES: ALGEBRA AND ALGORITHMS

FINAL REPORT

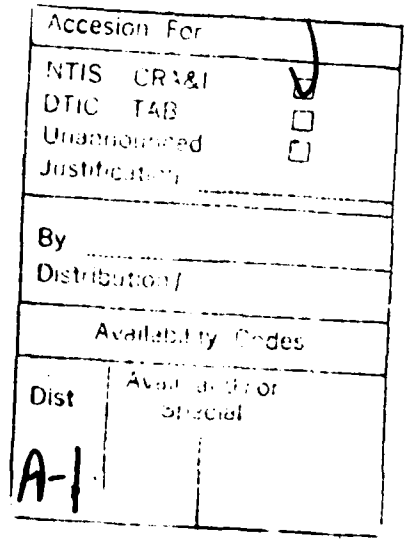
HARI KRISHNA

MARCH 31, 1990

U.S. ARMY RESEARCH OFFICE

CONTRACT NUMBER: DAAL03-87-K-0027

SYRACUSE UNIVERSITY



TOEPLITZ MATRICES: ALGEBRA AND ALGORITHMS

Hari Krishna

Department of Electrical & Computer Engineering

Syracuse University

Syracuse, NY, 13244-1240

(315) 443-1414

Contents

1 INTRODUCTION 1

2 On Fast Algorithm For Testing The Stability Of Discrete Time Systems 4

2.1 The Levinson Algorithm 5

2.2 The Classical Wide Sense and Strict Sense Stability 10

2.3 New Step-down Procedure 13

2.3.1 The Symmetric Case 13

2.3.2 The Anti-symmetric Case 14

2.3.3 The Singular Cases 16

2.3.4 Computational Complexity Consideraton 20

2.4 Further analysis of the sigular cases 24

3 Stability of Algorithm 27

3.1 Stability of Levinson Algorithm 28

3.2 Stability of Split Levinson Algorithm 31

3.3 Residual of Split Levinson Algorithm 35

3.4 The bound of residual 41

4 The Fast and Superfast Algorithms for Block Toeplitz Matries 47

4.1 The Fast Algorithm for Block Toeplitz Matrices 48

4.2 The Schur Algorithm for Block Toeplitz Matrices 57

4.3 The Superfast Algorithm for Block Toeplitz Matrices 61

4.4 Conclusion 68

5 The Finite Precision Analysis of Classical and Split Algorithms 69

5.1 The Classical Algorithms 70

5.1.1 The Levinson Algorithm 70

5.1.2 Schur Algorithm 71

5.1.3 Lattice Algorithm 73

5.2 Split Algorithms 75

5.2.1 Split Levinson Algorithm 75

5.2.2 Split Schur Algorithm 78

5.2.3 Split Lattice Algorithm 80

5.3 Finite Precision Analysis of Classical Algorithms 81

5.3.1 Finite Analysis of Levinson Algorithm 81

5.3.2 Finite Analysis of Schur Algorithm 86

5.3.3 Finite Analysis of Lattice Algorithm 90

5.4 Finite Analysis of Split Algorithm 93

5.4.1 Finite Analysis of Split Levinson Algorithm 93

5.4.2 Finite Analysis of Split Schur Algorithm 93

5.4.3 Finite Analysis of Split Lattice Algorithm 94

6 On Reduced Polynomial Based Split Algorithms **96**

6.1 Notation and Classical Algorithms 98

6.1.1 Notation 99

6.1.2 The Levinson-Durbin Algorithm 100

6.1.3 The Schur Algorithm 101

6.1.4 The Lattice Algorithm 102

6.1.5 The Normalized Lattice Algorithm 104

6.2 A New Split Levinson Algorithm 105

6.3 The Split Schur Algorithm 113

6.4 The Split Lattice Algorithms 114

6.5 Related Algorithms and Extensions 120

7 Conclusions **124**

Abstract

In this research project, we analyze the mathematical structure and numerical algorithms associated with Toeplitz matrices. Toeplitz matrices arise in a number of problems in engineering and applied mathematics. In many such problems, the task is to solve for certain parameters of interest (such as predictor polynomial, reflection coeffients, and solution to a linear system) in a computationally efficient manner. Also, the numerical stability aspects of the various algorithms must be examined from the standpoint of implementation using finite precision arithmetic. We have derived fast (order-recursive) and superfast (fast Fourier transform based) algorithms for solving a Toeplitz linear system. The algorithms reported here are some of the most computationally efficient algorithms. Also, the numerical stability of the split Levinson algorithm is examined and it is established that it is weakly stable. Furthermore the various classical and split Levinson algorithms are studied for the effects of finite precision arithmetic. An interesting relationship between Levinson algorithm and stability tests for discrete systems is exploited to derive a new computationally efficient algorithm for testing the wide sense stability of discrete time systems.

Chapter 1

INTRODUCTION

Toeplitz matrices occur in a large number of problems in engineering and applied mathematics. In problems, we have studied in pattern recognition and prediction and filtering theory, they arise as the autocovariance matrix of a wide sense stationary stochastic process. Some of the important computational problems associated with a Toeplitz matrix that we have studied are (i) computation of the associated predictor polynomial and reflection coefficients, and (ii) computation of the solution to a Toeplitz linear system. Emphasis is placed upon exploiting the rich mathematical structure of the Toeplitz matrix to derive algorithms having as low arithmetic complexity as possible.

Another important aspect of numerical algorithms is their stability when implemented using finite precision (either fixed point or floating point) arithmetic. Such an analysis is critical when the data is perturbed by errors and the underlying problem is ill-conditioned. The numerical stability of an algorithm determines its applicability for solving practical problems of interest.

In this research project, we have focused on computational aspects of linear algebra problems associated with Toeplitz matrices. The matrix structures considered are one dimensional as well as two dimentional. We have derived fast (order-recursive) as well as superfast (fast Fourier transform based) algorithms for solving a Toeplitz linear system. The algorithms reported here are some of the most computationally efficient algorithms for solving Toeplitz systems. Also, the numerical stability of the split Levinson algorithm is examined and it is established that it is weakly stable. Furthermore, the various classical and split algorithms are studied for the effects of finite precision arithmetic. An interesting relatoinship between the Levinson algorithm and stability tests for discrete systems is exploited to derive a new computationally efficient algorithm for testing the wide sense stability of discrete time systems.

The main contributions of this research project are as follows:

1. A new algorithm for testing the wide sense stability of discrete time systems is derived. A system is term stable in the wide sense if none of its poles are outside the unit circle. The new algorithm requires $0.25n^2$ multiplications (MULT) and $0.5n^2$ additions (ADD). This represents a reduction by 50% in the multiplicative complexity over previous algorithms.

2. The numerical stability of the split Levinson algorithm is studied and various bounds on the norm of the residual vector are derived. It leads to an important result - the split Levinson algorithm is weakly stable.

3. The block Toeplitz matrices are studied and new fast and superfast computationally

efficient algorithms are derived for solving a block Toeplitz linear system. Additional computational savings are realized when the block Toeplitz matrix has additional structure.

4. The effects of finite precision arithmetic on the computation of reflection coefficients using the classical as well as the new split algorithms, are studied. Various bounds are derived which demonstrate the propagation of errors from one stage to the next in the order recursive algorithm.

5. A new form of three-term recurrence relation is derived and computationally efficient alternatives to the Levinson, Schur, lattice and normalized lattice algorithms are obtained. We also describe a new lattice realization of a digital filter and a new computationally efficient algorithm for solving the Toeplitz linear system.

This report is organized as follows. In Chapter 2, the Levinson algorithm and its salient feature are described. Also, based on the Levinson algorithm, new tests are derived for testing the strict sense and wide sense stability of a discrete time system. In Chapter 3, the numerical stability of the split Levinson algorithm is studied. Chapter 4 deals with fast (order-recursive) and superfast (fast Fourier transform based) algorithms for solving block Toeplitz linear systems. Chapter 5 deals with the finite precision analysis of classical as well as split algorithms for computing the reflection coefficients. In Chapter 6, new algorithms are derived for computing the reflection coeffients and subsequently solving the Toeplitz linear system. Chapter 7 contains some conclusions for this research work.

Chapter 2

On Fast Algorithm For Testing The Stability Of Discrete Time Systems

Stability is a basic consideration in the design of any system. Usually , bounded output is required for bounded input. The most popular technique to test stability of a given system is to locate the root distribution of the characteristic function of the system. Several methods for the popuse such as Routh and Hurwitz stability criteria have been proposed. These methods, however, are complicated in computation because of their generality.

In this chapter, a fast and efficient algorithm for the testing stability will be studied. The main point is that instead of computing the roots, a set of parameter called the reflection cofficients are computed. A close relation between the reflection cofficients and roots of a given characteristic polynomial is that the roots are located inside the unit circle if and only if the magnitude of each of the reflection cofficients is less than one. The methods are based on the Levinson algorithm which is widely used in digital signal processing.

2.1 The Levinson Algorithm

The minimum least square method is used in the design of discrete systems in applications such as power spectrum estimation,and speech prediction. The linear systems obtained often have a special structure. This special system is called the Toeplitz system. There are efficient algorithms for solving the Toeplitz linear system because of its unique mathematical properties. Levinson algorithm, an efficient and fast method for solving the Toeplitz systems, was introduced by N. Levinson [1]. Toeplitz systems occur widely in modern signal processing.

Suppose that a stationary, finite energy signal sequence $u(l)$ is given. If we want to predict the $u(l)$ by its previous n data, we form

$$u(l) = \sum_{i=1}^{n} -a_i u(l-i) \tag{2.1}$$

where a_i, $i = 1, 2, \cdots, n$ are called the prediction coefficients and are determined by minimizing the square of prediction error

$$\alpha_n = \left(u(l) + \sum_{i=1}^{n} a_i u(l-i) \right)^2 \tag{2.2}$$

which results in the following linear system of equations,

$$T_{n-1} \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = - \begin{pmatrix} c_1 \\ \vdots \\ c_n \end{pmatrix} \tag{2.3}$$

where

$$T_{n-1} = \begin{pmatrix} 1 & c_1 & \cdots & c_{n-1} \\ c_1 & 1 & \cdots & c_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n-1} & c_{n-2} & \cdots & 1 \end{pmatrix} \tag{2.4}$$

is a Toeplitz matrix, and

$$c_i = \sum_{j=1}^{n} u(j)u(j-i) \tag{2.5}$$

The equation obtained is called also Yule-Walker equation or normal equation. Let

$$\underline{c}_n^t = (1\ c_1\ c_2\ \cdots\ c_n) \tag{2.6}$$

and

$$\underline{a}_n^t = (1\ a_1\ \cdots\ a_n) \tag{2.7}$$

then, the minimum error α_n is given by

$$\alpha_n = 1 + c_1a_1 + c_2a_2 + \cdots + c_na_n = \underline{c}_n^t\underline{a}_n \tag{2.8}$$

Combining (2.3) and (2.8), the normal equation can be written as

$$T_n\underline{a}_n = (\alpha_n\ 0\ \cdots\ 0)^t \tag{2.9}$$

For a given normal equation, one must find the inverse of the Toeplitz matrix to solve for the unknown vector $\underline{a}_n$. The classical algorithms for solving the linear system are Gaussian elemination and Cholesky decomposition. Both of them, however, are designed for an arbitrary matrix. The special structure of Toeplitz matrix is not exploited by these general

methods, that is, even though these classical methods can be used to solve the Toeplitz system in (2.9), they are not efficient. Generally speaking, the operation time required by Gaussian elemination and Cholesky decomposition is proportional to n^3. The efficient algorithm which exploits the structure of the Toeplitz matrix should have smaller order.

One of the efficient algorithm for solving Toeplitz problem is Levinson algorithm. It is widely used for its simple and efficient structure. The general Levinson algorithm will be studied in Chapter 3. The details are omited here.

The Levinson algorithm has a form

$$\rho_k \alpha_{k-1} = \sum_{i=0}^{k-1} c_{k-i} a_{k-1,i} \tag{2.10}$$

$$a_{k,i} = a_{k-1,i} + \rho_k a_{k-1,k-i} \tag{2.11}$$

$$\alpha_k = \alpha_{k-1}(1 - \rho_k^2) \tag{2.12}$$

with initial conditions $a_{k,0} = 1$, $a_{k,k} = \rho_k$ and $\alpha_0 = 1$.

In this report, the polynomial representation of a sequense is also used. For a given vector $\underline{a}_k = (a_{k,0} \ \cdots \ a_{k,k})^t$, the polynomial form is defined as

$$a_k(z) = \sum_{i=0}^{k} a_{k,i} z^{-i} \tag{2.13}$$

The Levinson algorithm in polynomial form is

$$a_k(z) = a_{k-1}(z) + \rho_k z^{-1} \hat{a}_{k-1}(z) \tag{2.14}$$

where $\hat{a}_{k-1}(z)$ denotes the reciprocal polynomial of $a_{k-1}(z)$ and is defined as

$$\hat{a}_k(z) = z^{-k} a_k(z^{-1}) \tag{2.15}$$

After getting the $a_n(z)$ by Levinson algorithm, the inverse matrix of Toeplitz matrix T_n can be obtained through the Gohberg-Semencul formula

$$
\begin{aligned}
T_n^{-1} \;=\; & \alpha_n^{-1} \begin{bmatrix} 1 & 0 & \cdots & 0 \\ a_{n,1} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,n} & \cdots & a_{n,1} & 1 \end{bmatrix} \begin{bmatrix} 1 & a_{n,1} & \cdots & a_{n,n} \\ 0 & 1 & \cdots & a_{n,n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \\
& -\alpha_n^{-1} \begin{bmatrix} 0 & 0 & \cdots & 0 \\ a_{n,n} & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,n} & 0 \end{bmatrix} \begin{bmatrix} 0 & a_{n,n} & \cdots & a_{n,1} \\ 0 & 0 & \cdots & a_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix}
\end{aligned} \tag{2.16}
$$

The general formula will be derived in next chapter.

Let us define a new matrix A_n as

$$
A_n = \begin{bmatrix} 1 & a_{1,1} & a_{2,2} & \cdots & a_{n,n} \\ 0 & 1 & a_{2,1} & \cdots & a_{n,n-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \tag{2.17}
$$

In the following we show that

$$
T_n^{-1} = A_n \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & \alpha_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_n \end{bmatrix} A_n^t \tag{2.18}
$$

Proof: Since we have

$$T_nA_n = \begin{pmatrix} T_n \begin{pmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & T_n \begin{pmatrix} a_{1,1} \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} & \cdots & T_n \begin{pmatrix} a_{n,n} \\ a_{n,n-1} \\ \vdots \\ a_{n,1} \\ 1 \end{pmatrix} \end{pmatrix}$$

using (2.9), we have

$$T_nA_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ * & \alpha_1 & 0 & \cdots & 0 \\ * & * & \alpha_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & \alpha_n \end{pmatrix}$$

$$= \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ * & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ * & * & * & \cdots & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \alpha_n \end{pmatrix}$$

where the stars represent the terms that we do not care about. Multipling both side by A_n^{-1} we have

$$T_n = R_nD_nA_n^{-1}$$

since T_n is symmetric, we have

$$T_n^t = A_n^{-t}D_nR_n^t = T_n$$

comparing the results, we see that $R_n = A_n^{-t}$. The proof is complete.

This result gets the Cholesky decompositon of inverse Toeplitz matrix. The results (2.16) and (2.17) are different. The matrices in (2.16) are also Toeplitz. Based on the structure, the displacement ranks of matrices which can be represented by a sum of two product Toeplitz matrix is studied by [2].

2.2 The Classical Wide Sense and Strict Sense Stability

The Levinson algorithm introduced in last section can be used in application such as linear predictive coding and spectral analysis. It is also used in design of ARMA filter for testing stability of the filter.

Stability is a basic consideration in the design of any system. For a discrete time system, one needs to check the location of the roots of the charecteristic polynomial with repect to the unit circle (UC) for testing the stability of the system [22, 23]. The concept of stability has been extended in [3]. The system is said to be stable in the strict sense, if all the roots are inside the UC. Strict sense stability (SSS) is same as the classical definition of stability and can be found in any standard text book on signal processing. The readers are referred to [22, 23]. The system is said to be stable in the wide sense, if no roots are outside the UC (all the roots lie inside or on the UC). The wide sense stability (WSS) is a new idea. It was proposed after a study of the spectral analysis of ARMA filters and adaptive filters [4, 24]. In the following, we will discuss a algorithm for testing stability. It

is based on the Levinson algorithm. After that we will present a new algorithm for testing the WSS.

Let the characteristic polynomial of a given filter be

$$a_n(z) = \sum_{i=0}^{n} a_{n,i} z^{-1} \tag{2.19}$$

it is assumed, without any loss of generality, that the first coefficient of $a_n(z)$ is unity.

From the previous chapter, the Levinson algorithm can be written as

$$a_k(z) = a_{k-1}(z) + \rho_k z^{-1} \hat{a}_{k-1}(z) \tag{2.20}$$

$$\rho_k = a_{k,k} \tag{2.21}$$

The reciprocal polynomial of $a_k(z)$ is

$$\hat{a}_k(z) = z^{-1} \hat{a}_{k-1}(z) + \rho_k a_{k-1}(z) \tag{2.22}$$

Solving (2.20) and (2.22) for $a_{k-1}(z)$ leads to

$$a_{k-1}(z) = \frac{a_k(z) - \rho_k \hat{a}_k(z)}{1 - \rho_k^2} \tag{2.23}$$

Equations (2.21) and (2.23) constitute the inverse Levinson algorithm. For a given $a_n(z)$, the reflection coefficients ρ_k, $k = n, \cdots, 1$, can be computed using (2.21) and (2.23). It has been proved that the roots of $a_n(z)$ are located inside the UC, if and only if (iff) the reflection coefficients satisfy

$$|\rho_k| < 1, \qquad k = 1, 2, \cdots, n \tag{2.24}$$

So for testing the strict sense stability of a system, we first compute the reflection coefficients using the inverse Levinson algorithm. If the system is stable, the condition (2.24) should be satisfied. This procedure is also called the stepdown procedure [25].

It is clear from (2.23) that if $|\rho_k| = 1$, or if some roots are located on the unit circle, we can not use the classical stepdown procedure to compute $a_{k-1}(z)$. For the SSS, we claim that the system is unstable in this situation.

It is established in [3] that if $\rho_k = 1$, then $a_k(z)$ must satisfy

$$a_k(z) = \hat{a}_k(z) \tag{2.25}$$

and if $\rho_k = -1$, then $a_k(z)$ must satisfy

$$a_k(z) = -\hat{a}_k(z) \tag{2.26}$$

for wide sense stability. The polynomials which satisfy (2.25) are called symmetric. The polynomials which satisfy (2.26) are called anti-symmetric.

If $a_k(z)$ is symmetric or anti-symmetric, then it is proved in [3] that $a_{k-1}(z)$ can be expressed as

$$\hat{a}_{k-1}(z) = \frac{1}{k}\frac{d\hat{a}_k(z)}{dz^{-1}} \tag{2.27}$$

or computed as

$$a_{k-1}(z) = 1 + \frac{1}{k}\sum_{i=1}^{k-1}(k-i)a_{k,i}z^{-i} \tag{2.28}$$

At this point, we have overcome the problem caused by $|\rho_k| = 1$ in classical step-down procedure.

This result leads to a test for WSS. It can be summarized as:" the necessary and sufficient condition for WSS are $|\rho_k| \leq 1, k = 1, 2, \cdots n$ (necessary condition for WSS) and if $|\rho_k| = 1$, then the corresponding polynomial $a_k(z)$ must be either symmetric or anti-symmetric." We should note that the symmetric or anti-symmetric polynomials are specified in term

of approximate by one-half of their coefficients. In the following section, we discuss the tests for SSS and WSS based on the recursive computation of symmetric or anti-symmetric polynomials.

2.3 New Step-down Procedure

In the last section, we discussed the classical Levinson algorithm for testing the WSS and SSS. A new fast algorithm is introduced in this section. This algorithm is called the step-down precedure. It is based on the recursive computation of either symmetric or anti-symmetric polynomials. We discuss the symmetric case first.

2.3.1 The Symmetric Case

The new step-down procedure is based on the recursive computation of the symmetric polynomial $s_k(z)$ defined as

$$s_k(z) = a_{k-1}(z) + z^{-1}\hat{a}_{k-1}(z) \tag{2.29}$$

It is clear from (2.29) that $s_{k,0} = s_{k,k} = a_{k-1,0} = 1$, $k = 1, 2, \cdots n$. From (2.23), we have

$$\begin{aligned} \hat{a}_{k-1}(z) &= \frac{z^{-(k-1)}[a_k(z^{-1}) - \rho_k \hat{a}_k(z^{-1})]}{1-\rho_k^2} \\ &= \frac{z[\hat{a}_k(z) - \rho_k a_k(z)]}{1-\rho_k^2} \end{aligned}$$

Dividing both sides by z and adding the result with (2.23), we have

$$s_k(z) = \frac{a_k(z) + \hat{a}_k(z)}{1+\rho_k} \tag{2.30}$$

Now we derive the new step-down procedure [26, 5]. Using (2.30), we can write

$$(1+\rho_k)s_{k-1}(z) = a_{k-1}(z) + \hat{a}_{k-1}(z)$$

Using this equation with (2.30), we have

$$a_{k-1}(z) = \frac{s_k(z) - z^{-1}s_{k-1}(z)(1+\rho_{k-1})}{1-z^{-1}}$$

and

$$\hat{a}_{k-1}(z) = \frac{s_{k-1}(z)(1+\rho_{k-1}) - s_k(z)}{1-z^{-1}}$$

Substituting them into (2.23) and rearranging the terms, we have

$$s_{k-1}(z) = \alpha_k^{-1} z[(1+z^{-1})s_k(z) - s_{k+1}(z)] \tag{2.31}$$

where

$$\alpha_k = 1 + s_{k,1} - s_{k+1,1} \tag{2.32}$$

and

$$\rho_{k-1} = -1 + \frac{\alpha_k}{1-\rho_k} \tag{2.33}$$

In the above, (2.32) is obtained by noting that $s_{k-1,0} = 1$. Given the polynomial $a_n(z)$, the initial polynomials $s_{k+1}(z)$ and $s_n(z)$ are obtained from (2.29) and (2.30) by setting $k = n+1$ and $k = n$ respectively,. where ρ_n is obtained from $\rho_n = a_{n,n}$.

2.3.2 The Anti-symmetric Case

In this case, the new step-down procedure is based on the recursive computation of the anti-symmetric polynomial $t_k(z)$ defined as

$$t_k(z) = a_{k-1}(z) - z^{-1}\hat{a}_{k-1}(z) \tag{2.34}$$

Once again, we have $t_{k,0} = -t_{k,k} = 1$, $k = 1, 2, \cdots n$. The derivation is very similar to the symmetric case and the various expressions can be summarized as follows

$$t_k(z) = \frac{a_k(z) - \hat{a}_k(z)}{1 - \rho_k} \tag{2.35}$$

$$t_{k-1}(z) = \beta_k^{-1} z[(1 + z^{-1})t_k(z) - t_{k+1}(z)] \tag{2.36}$$

where

$$\beta_k = 1 + t_{k,1} - t_{k+1,1} \tag{2.37}$$

and

$$\rho_{k-1} = 1 - \frac{\beta_k}{1 + \rho_k} \tag{2.38}$$

Given the polynomial $a_n(z)$, the initial polynomials $t_{n+1}(z)$ and $t_n(z)$ are obtained from (2.34) and (2.35) by setting $k = n + 1$ and $k = n$ respectively, where ρ_n is obtained from $\rho_n = a_{n,n}$. Since either one of (2.31)–(2.33) or (2.35)–(2.38) may be employed for the reflection coefficient computation, they may also be used as the test for SSS. The reflection coefficients are computed using either (2.31)–(2.33) or (2.35)–(2.38) and if $|\rho_k| < 1$, $k = 1, 2, \cdots n$, then the system is stable in the strict sense.

We will give an example to describe the procedure of the algorithms. We use the symmetric case, the anti-symmetric case is almost the same as symmetric case.

Example 1. Let Z^n denote the vector $Z^n = (1\ z^{-1}\ \cdots\ z^{-n})^t$. Consider the transfer function $a_4(z) = (1\ 1.6\ 0.11\ -0.844\ -0.336)Z^4$. Here, $n = 4$, and $\rho_4 = -0.336$. We use (2.39) and (2.40) to compute $s_5(z)$ as, $s_5(z) = (1\ 1.264\ -0.734\ -0.734\ 1.264\ 1)Z^5$ and $s_4(z) = (1\ 1.1386\ 0.3313\ 1.386\ 1)Z^4$. Employing the recurrence in (2.31)–(2.33), we obtain

$$\alpha_4 = 0.8746$$

$$\rho_3 = 0.3454$$

$$s_3(z) = (1\ 2.52\ 2.52\ 1)Z^3$$

$$\alpha_3 = 2.3814$$

$$\rho_2 = 0.7701$$

$$s_2(z) = (1\ 1.9773\ 1)Z^2$$

$$\alpha_2 = 0.4573$$

$$\rho_1 = 0.9890$$

$$s_1(z) = (1\ 1)Z$$

Since, $|\rho_k| < 1$, $k = 1, 2, 3, 4$, the polynomial is stable in the strict sense. The roots of $a_4(z)$ are $z_1 = 0.2$, $z_2 = -0.3$, $z_3 = -0.7$, $z_4 = -0.8$. They are all inside the unit circle.

We observe that if one of the reflection coefficients, say ρ_k is such that $|\rho_k| = 1$, then the step-down procedures can not be used for the reflection coefficient computation. In the next section, we derive alternative techniques to incorporate such cases referred to as the singular cases.

2.3.3 The Singular Cases

The condition $|\rho_k| \leq 1$ is only a necessary but not a sufficient condition for WSS. In addition, the new step-down procedures as outlined in last section can not used if $|\rho_k| = 1$. In this section, we derive modifications to the new step-down procedures to incorporate the case $|\rho_k| = 1$. This implies that either $\rho_k = 1$ or $\rho_k = -1$. In the following, we discuss the cases $\rho_k = -1$ and $\rho_k = 1$ separately. We begin our analysis with the case $\rho_k = -1$.

i). $\rho_k = -1$

In this case, for testing the WSS, we need to check if $a_k(z)$ is anti-symmetric or not. The following lemmas play a key role for such a check.

Lemma 1 *Whenever $\rho_k = -1$ and $|\rho_{k+1}| \neq 1$, the polynomial $a_k(z)$ is anti-symmetric iff*

$$(1 + z^{-1})s_{k+1}(z) - s_{k+2}(z) = 0 \tag{2.39}$$

Proof: Iterating (2.31) one step forward, we have

$$s_k(z) = \alpha_{k+1}^{-1} z[(1 + z^{-1})s_{k+1}(z) - s_{k+2}(z)]$$

subsituting $s_k(z)$, in terms of $a_k(z)$ and $\hat{a}_k(z)$, using (2.28), we obtain

$$a_k(z) + \hat{a}_k(z) = z(1 - \rho_{k+1})^{-1}[(1 + z^{-1})s_{k+1} - s_{k+2}(z)]$$

Using the above expression, the statement of the lemma follows in a straightforward manner. Also if $a_k(z) = -\hat{a}_k(z)$, then using (2.29), we get

$$a_k(z) = \frac{s_{k+1}(z)}{1 - z^{-1}} \tag{2.40}$$

Lemma 2 *Whenever $\rho_k = -1$ and $|\rho_{k+1}| \neq 1$, the polynomial $a_k(z)$ is anti-symmetric iff*

$$(1 + z^{-1})t_k(z) - t_{k+1}(z) = 0 \tag{2.41}$$

Proof: Using (2.34) and (2.35), we have

$$\begin{aligned} t_{k+1}(z) &= a_k(z) - z^{-1}\hat{a}_k(z) \\ t_k(z) &= \frac{a_k(z) - \hat{a}_k(z)}{1 - \rho_k} \end{aligned}$$

Solving for $a_k(z)$ and $\hat{a}_k(z)$, we have

$$\begin{aligned} a_k(z) &= (1-z^{-1})^{-1}[t_{k+1}(z) - zt_k(z)(1-\rho_k)] \\ \hat{a}_k(z) &= (1-z^{-1})^{-1}[t_{k+1}(z) - t_k(z)(1-\rho_k)] \end{aligned}$$

Adding them and setting $\rho_k = -1$, we have

$$a_k(z) + \hat{a}_k(z) = -2(1-z^{-1})^{-1}[(1+z^{-1})t_k(z) - t_{k+1}(z)]$$

and the lemma follows.

Also, if $a_k(z) = -\hat{a}_k(z)$, then using (2.35), we obtain

$$a_k(z) = t_k(z) \tag{2.42}$$

ii). $\rho_k = 1$

In this case, for testing the WSS, we need to check if $a_k(z)$ is symmetric or not. The following lemmas play a key role for such a check.

Lemma 3 *Whenever $\rho_k = 1$ and $|\rho_{k+1}| \neq 1$, the polynomial $a_k(z)$ is symmetric iff,*

$$(1+z^{-1})s_k(z) - s_{k+1}(z) = 0 \tag{2.43}$$

If (2.43) holds, then we also have

$$a_k(z) = s_k(z) \tag{2.44}$$

Lemma 4 *Whenever $\rho_k = 1$ and $|\rho_{k+1}| \neq 1$, the polynomial $a_k(z)$ is symmetric iff,*

$$(1+z^{-1})t_{k+1}(z) - t_{k+2}(z) = 0 \tag{2.45}$$

If (2.45) holds, then we also have

$$a_k(z) = (1 - z^{-1})^{-1} t_{k+1}(z) \tag{2.46}$$

The proofs for Lemma 3 and Lemma 4 are similar to the proofs for Lemma 2 and Lemma 1 respectively. On the basis of the above analysis and the test for WSS using $a_k(z)$, the tests for WSS that employ the polynomials $s_k(z)$ or $t_k(z)$ can be described in a straightforward manner. In the following, we summarize the test for WSS using $s_k(z)$. The test for WSS using $t_k(z)$ may be described in a similar manner.

Test for WSS using $s_k(z)$

Step 1. Use the stepdown procedure given in (2.31)-(2.33) and compute ρ_k, $k = n, n-1, \cdots$

If $|\rho_k| > 1$, then declare " unstable " and stop.

If $\rho_k = -1$, Go to step 2. If $\rho_k = 1$, Go to step 3.

Step 2. Check if (2.39) is satisfied.

If not, then declare " unstable " and stop.

If yes, compute $a_k(z)$ and $a_{k-1}(z)$ using (2.40) and (2.28) respectively. Go to Step 1.

Step 3. Check if (2.43) is satisfied.

If not, then declare " unstable " and stop.

If yes, then obtain $a_k(z)$ and $a_{k-1}(z)$ unsing (2.44) and (2.28) respectively. Go to Step 1.

2.3.4 Computational Complexity Consideraton

Using the symmetry properties of the $s_k(z)$, it may be established that if none of the reflection coefficients have unit magnitude, then the new step-down procedure requires $0.25n^2 + n$ MULT and $0.5n^2 + 4n$ ADD. If one of the reflection coefficients, say $\rho_k = -1$, then it requires an additional $2k$ MULT and $0.5k$ ADD. Similarly, additional $2k$ MULT and $0.5k$ ADD are required if $\rho_k = 1$. Similarly, using the symmetry properties of the $t_k(z)$, it may be established that if none of the reflection coefficients have unit magnitude, then the step-down procedure requires $0.25n^2 + 0.5n$ MULT and $0.5n^2 + 3n$ ADD. If $\rho_k = -1$, then it requires an additional k MULT and $0.5k$ ADD. If $\rho_k = 1$, then it requires an additional $2k$ MULT and $1.5k$ ADD. Thus, the procedures developed here require one-half the number of MULT and the same number of ADD as the procedures described in last section. The computational savings in this work stem basically from a more efficient computation of reflection coefficients. Also, the computational complexity of the algorithms presented here is the same as the algorithms presented in [27]. However, the methodology and the technique for testing the stability in the algorithms presented here and those in [27] are entirely different. In the following, we give several examples to illustrate the algorithm.

Example 2. Let $a_4(z) = (1\ 0.4\ 0.48\ 0.68\ -0.4)Z^4$. Here, $n = 4$, and $\rho_4 = -0.4$. We use (2.29) and (2.30) to compute $s_5(z)$ and $s_4(z)$ as

$$s_5(z) = (1\ 0\ 1.16\ 1.16\ 0\ 1)Z^5$$

$$s_4(z) = (1\ 1.8\ 1.6\ 1.8\ 1)Z^4$$

Employing the recurrence in (2.31)–(2.33), we obtain

$$\begin{aligned} \alpha_4 &= 2.8 \\ \rho_3 &= 1 \\ s_3(z) &= (1\ 0.8\ 0.8\ 1)Z^3 \end{aligned}$$

Since $\rho_3 = 1$, we check that Lemma 3 is satisfied. Thus $a_3(z) = s_3(z)$, and using (2.28), we obtain

$$\begin{aligned} a_2(z) &= (1\ \frac{1.6}{3}\ \frac{0.8}{3})Z^2 \\ \rho_2 &= \frac{0.8}{3} \\ s_2(z) &= (1\ \frac{16}{19}\ 1)Z^2 \end{aligned}$$

Using (2.31)–(2.33) once again, we obtain

$$\begin{aligned} \alpha_2 &= \frac{19.8}{19} \\ \rho_1 &= \frac{8}{19} \\ s_1(z) &= (1\ 1)Z \end{aligned}$$

Since $|\rho_k| \leq 1$, $k = 1, 2, 3, 4$ and Lemma 3 is satisfied for $\rho_3 = 1$, $a_4(z)$ is stable in wide sense. The roots of $a_4(z)$ are $z_1 = 0.4, z_2 = -1$, $z_3, z_4 = -0.1 \pm 0.99498j$ and the magnitudes of z_2, z_3 and z_4 are one.

Example 3. Let

$$a_4(z) = (1\ 0.5\ -1.04\ -0.76\ 0.3)Z^4$$

Here, $n = 4$ and $\rho_4 = 0.3$. We use (2.29) and (2.30) to compute $s_5(z)$ and $s_4(z)$ as

$$\begin{aligned} s_5(z) &= (1\ 0.8\ -1.8\ -1.8\ 0.8\ 1)Z^5 \\ s_4(z) &= (1\ -0.2\ -1.6\ -0.2\ 1)Z^4 \end{aligned}$$

In this case, $\alpha_4 = 0$ implying that $\rho_3 = -1$. We check that Lemma 1 is satisfied. Obtain $a_3(z)$ using (2.40) as

$$a_3(z) = \frac{s_4(z)}{1 - z^{-1}} = (1\ 0.8\ -0.8\ -1)Z^3$$

and obtain $a_2(z)$ using (2.34) as

$$a_2(z) = (1\ \frac{1.6}{3}\ -\frac{0.8}{3})Z^2$$

Finally,

$$\begin{aligned} \alpha_3 &= \frac{4.4}{3} \\ \rho_2 &= -\frac{0.8}{3} \\ s_2(z) &= (1\ \frac{16}{11}\ 1)Z^2 \\ \alpha_2 &= \frac{72.2}{33} \\ \rho_1 &= \frac{8}{11} \\ s_1(z) &= (1\ 1)Z \end{aligned}$$

Once again, the conditions for WSS are satisfied. The roots of $a_4(z)$ are $z_1 = 0.3$, $z_2 = 1$, $z_3, z_4 = 0.9 \pm 0.43589j$. Note that $|z_2| = |z_3| = |z_4| = 1$.

Example 4. Let

$$a_5(z) = (1\ 1.3\ -2.6\ -1.9\ 1.4\ 0.8)Z^5$$

Here, $n = 5$, and $\rho_5 = 0.8$. We use (2.29) and (2.30) to compute $s_6(z)$ and $s_5(z)$ as,

$$\begin{aligned} s_6(z) &= (1\ 2.1\ -1.2\ -3.8\ -1.2\ 2.1\ 1)Z^6 \\ s_5(z) &= (1\ 1.5\ -2.5\ -2.5\ 1.5\ 1)Z^5 \end{aligned}$$

Employing the recurrence in (2.31)-(2.33), we obtain

$$\begin{aligned} \alpha_5 &= 0.4 \\ \rho_4 &= 1 \\ s_4(z) &= (1\ 0.5\ -3\ 0.5\ 1)Z^4 \end{aligned}$$

Since $\rho_4 = 1$, we check that Lemma 3 is satisfied. Thus, $a_4(z) = s_4(z)$ and using (2.34), we obtain

$$\begin{aligned} a_3(z) &= (1\ \frac{3}{8}\ -1.5\ \frac{1}{8})Z^3 \\ \rho_3 &= \frac{1}{8} \\ s_3(z) &= (1\ -1\ -1\ 1)Z^3 \end{aligned}$$

Using (2.31)-(2.33), once again, we obtain

$$\begin{aligned} \alpha_3 &= -0.5 \\ \rho_2 &= -\frac{11}{7} \\ s_2(z) &= (1\ -2\ 1)Z^2 \end{aligned}$$

Now, $\alpha_2 = 0$ imply $\rho_1 = -1$. We check that Lemma 1 is satisfied, and obtain $a_1(z)$ as

$$a_1(z) = \frac{s_2(z)}{1 - z^{-1}} = (1\ -1)Z$$

Finally, $s_1(z) = (1\ 1)Z$.

Since, $|\rho_2| = \frac{11}{7} > 1$, the conditions for WSS are not satisfied and $a_5(z)$ is unstable. The roots of $a_5(z)$ are $z_1 = 1$, $z_2 = 1$, $z_3 = 0.5$, $z_4 = -2$, $z_5 = 0.8$.

2.4 Further analysis of the sigular cases

Here, we analyze the singular cases in terms of roots of the polynomial $a_n(z)$ that lie on unit circle or occur in reciprocal pairs. Let u_1, u_2, $\cdots$, u_l be the roots on unit circle, that is $|u_j| = 1$, $j = 1, 2, \cdots, l$ and r_1, r_2, $\cdots$, r_m be the roots occuring in reciprocal pairs. Also, let $u_{l+m}(z)$ be a polynomial of degree $l + m$ having u_1, u_2, $\cdots$, u_l, and r_1, r_2, $\cdots$, r_m as its roots, such that $u_{l+m,0} = 1$. Clearly, $u_{l+m}(z)$ is a factor of $a_n(z)$, and we can write

$$a_n(z) = u_{l+m}(z) r_{n-l-m}(z) \tag{2.47}$$

It is straightforward to verify that if 1 is a root of $u_{l+m}(z)$ of even multiplicity, then $u_{l+m}(z)$ is a symmetric polynomial, and we have

$$\hat{a}_n(z) = u_{l+m}(z) \hat{r}_{n-l-m}(z) \tag{2.48}$$

Similarly, if 1 is a root of $u_{l+m}(z)$ of odd multiplicity, then $u_{l+m}(z)$ is an anti-symmetric polynomial, and we have

$$\hat{a}_n(z) = -u_{l+m}(z) \hat{r}_{n-l-m}(z) \tag{2.49}$$

In either case, $u_{l+m}(z)$ is a factor of $\hat{a}_n(z)$ also, therby implying from (2.23) that $u_{l+m}(z)$ is a factor of $a_{n-1}(z)$ and so on. Therefore, we obtain

$$a_{l+m}(z) = u_{l+m}(z) \tag{2.50}$$

Also, $\rho_{l+m} = 1$ if $a_{l+m}(z)$ is symmetric and $\rho_{l+m} = -1$ if $a_{l+m}(z)$ is anti-symmetric. In either case, $|\rho_{l+m}| = 1$. Consequently, the singular cases detect and check for the presence of roots on the unit circle and roots occuring in reciprocal pairs.

Using (2.47), (2.48) and the definition of $s_{n+1}(z)$ and $s_n(z)$ in (2.29) and (2.30), we see that $u_{l+m}(z)$ is a factor of $s_{n+1}(z)$ and $s_n(z)$. Therefore, (2.31) implies that it is also of $s_{n-1}(z)$ and so on. Consequently, from (2.29), we obtain

$$s_{l+m}(z) = u_{l+m}(z) = a_{l+m}(z) \tag{2.51}$$

if $a_{l+m}(z)$ is symmetric, or

$$\begin{aligned} s_{l+m+1}(z) &= (1 - z^{-1})u_{l+m}(z) \\ &= (1 - z^{-1})a_{l+m}(z) \end{aligned} \tag{2.52}$$

if $a_{l+m}(z)$ is anti-symmetric.

Similar statements hold for polynomials $t_k(z)$, $k = n, n-1, \cdots, l$. It is interesting to note here that the singular cases will occur as many times as the largest multiplicity of a root on the unit circle or a root occurring in reciprocal pairs [27]. This is due to the fact that a root of a_k of multiplicity *gamma* ($\gamma > 1$) is also a root of $\frac{da_k(z)}{dz^{-1}}$ of multiplicity $\gamma - 1$.

Numerical properties (in terms of stability and sensitivity) of the algorithms described here as compared to their classical forms are an important issue and needs further unvestigation. However, such an analysis goes beyond the scope of this work. Finally, we end this section by stating that computationally efficient tests for SSS and WSS based on the stepdown procedure may also be described for polynomials with complex coefficients. The details of such an analysis will be presented elsewhere.

In this chaper, we introduced the Levinson algorithm for solving Toeplitz systems. We also studied the computationally efficient procedures for testing the wide sense stability and strict sense stability for polynomials with real coefficients. The procedures are based on efficient procedures for computing the associated reflection coefficients.

Chapter 3

Stability of Algorithm

The algorithms for solving linear system usually involve a lot of computations. The precision associated with the initial values of problem is limited, and therefore, that is, errors are inevitable. For some problem, the algorithms give incorrect results even though the errors in the initial values are very small. This problem has attracted a great deal of attention. In fact, an algorithm is termed stable if it gives correct results within the required precision range for small errors in the initial values. If an algorithm is stable, for small errors, one can guarantee that the result also has small errors.

In this chapter, we explore the numerical stability properties of the recently reported split Levinson algorithm for the computing the predictor polynomial associated with a positive-definite real symmetric Toeplitz matrix. Various bounds on the residual vector are derived for the fixed-point and floating-point implementation of the algorithm. These bounds are similar in form to the bounds derived by Cybenko for the Levinson algorithm and are obtained by converting a three-term recurrence for the error vector to an equivalent

two-term recurrence. The main conclusion of the paper is that the split Levinson algorithm is weakly stable.

3.1 Stability of Levinson Algorithm

In the chaper 2, we have studied the Levinson algorithm. It is widely used in the signal processing applications such as speech processing, estimation of power spectrum and so on. However, the stability properties of the algorithm are not completely known. On one hand, the theoretical results showed that the algorithm is sensitive to errors [13] while on the other hand, the practical results showed that the algorithm gives accurate results [14]. Recently, the stability of Levinson algorithm (it is at least as stable Cholesky method) was proved. This proof ended the controversy about stability of Levinson algorithm. It is shown that the algorithm is ill-conditioned when ρ_k, the reflection coefficients, are near the unit circle. In this case, the algorithm is far more sensitive to errors. The analysis, strictly speeking, falls in the category called the weak stability analysis. The concept of weak stability was introduced in [7]. According to Buch, the result in [6] showed that Levinson algorithm is weakly stable.

The proof of strictly stability of Levinson algorithm is still lacking. Since errors of the algorithm are intricately interleaved, they are very difficult to manipulate. In this section, we will study stability aspects of Levinson algorithm.

In Chapter 2, the Levinson algorithm was studied. For convenience, we write the algo-

rithm in the vector form as follows,

$$\rho_i = -\frac{c_i + \sum_{j=1}^{i-1} a_{i-1,j} c_{i-j}}{\alpha_{i-1}} \tag{3.1}$$

$$\underline{a}_i = \begin{pmatrix} \underline{a}_{i-1} + \rho_i \hat{\underline{a}}_{i-1} \end{pmatrix} \tag{3.2}$$

$$\alpha_i = \left(1 - \rho_i^2\right) \alpha_{i-1}, \quad i = 1, 2, \cdots, n \tag{3.3}$$

The initial condition is $\alpha_0 = 1$. Here, ρ_i is the called reflection coefficient, and the notation $\hat{}$ represents the operation of symmetric reflection of $\underline{a}_i$, i.e.

$$\hat{\underline{a}}_i = (a_{i,i} \; a_{i,i-1} \; \cdots \; a_1)$$

The necessary and sufficient condition for T_n to be positive definite is $|\rho_i| < 1$. This also guarantees that the IIR filter, whose characteristic polynomial is $1 + \sum_{j=1}^{i} a_{i,j} z^{-j}$, is stable. The error α_i decreases monotonically to zero as i is increased.

It is well known that the condition of matrix A is very important in solving $A\underline{x} = \underline{b}$. The error of $\underline{x}$ can be bounded as

$$\frac{\|\underline{x} - \underline{x}'\|}{\|\underline{x}\|} \leq \frac{k(A)\|E\|}{(1 - k(A)\frac{\|E\|}{\|A\|})\|A\|} + \frac{k(A)\|\underline{e}\|}{(1 - k(A)\frac{\|E\|}{\|A\|})\|\underline{b}\|} \tag{3.4}$$

where $\underline{x}'$ represents the solution obtained by a computer. In this work, we assume that the matrix A is correct, i.e. $\|E\| = 0$ and we use the maximum column sum matrix norm. The reader is referred to [15] for a discussion of the norm. Under these assumptions,(3.4) becomes

$$\frac{\|\underline{x} - \underline{x}'\|}{\|\underline{x}\|} \leq k(A)\frac{\|\underline{e}\|}{\|\underline{b}\|} \tag{3.5}$$

where $\underline{e}$ is residual vector.

For the Toeplitz matrix T_n, Cybenko has obtained the bound on $\|T_n^{-1}\|$ as [Cybenko, 1980],

$$max\left\{\frac{1}{\alpha_{n-1}}, \frac{1}{\sum_{j=1}^{n-1}(1-\rho_j)}\right\} \leq \|T_n^{-1}\| \leq \prod_{j=1}^{n-1}\frac{1+|\rho_j|}{1-|\rho_j|}. \tag{3.6}$$

Since the $|c_i| \leq 1$, the norm of T_n satisfies $1 \leq \|T_n\| \leq n$, and the bound on condition of Toeplitz matrix T_n is given by

$$max\left\{\frac{1}{\alpha_{n-1}}, \frac{1}{\prod_{j=1}^{n-1}(1-\rho_j)}\right\} \leq k(T_n) \leq n\prod_{j=1}^{n-1}\frac{1+|\alpha_j|}{1-|\alpha_j|}. \tag{3.7}$$

This is a very important result. It reveals that Yule-Walker equation is ill-conditioned for $|\rho_i|$ near one or for large n. If this happens, one must be very careful in the computations. The result is very sensitive to error. That is why Box and Jenkins warned that the algorithm is sensitive to error when n is large [13].

Cybenko has obtained bound on the norm of residual for Levinson algorithm, which are,

$$\|\delta_n\| \leq \Delta(\frac{n^3}{3} + \frac{3n^2}{2} + n)\prod_{j=1}^{n}(1+|\rho_j|) + O(\Delta^2) \tag{3.8}$$

if fixed-point arithmetic is used, and

$$\|\delta_n\| \leq \Delta(\frac{n^2}{2} + 11n)\left(\prod_{j=1}^{n}(1+|\rho_j|) - 1\right) + O(\Delta^2) \tag{3.9}$$

if floating-point arithmetic is used. These results tell us that Levinson algorithm gives a small residual if n is not large and Δ is small. In other words, Levinson algorithm is weakly stable as pointed out by Bunch [7]. The implication is that the residual is small even though $|\rho_i|$ is near to one if n and Δ are properly chosen. As a result, engineers use Levinson algorithm in speech processing and obtain reasonable results.

Markel and Gray have simulated the Levinson algorithm in the fixed-point arithmetic [14]. Their results support our argument here. Their results showed that for minimum wordlength, it is necessary to use preemphasis and a sampling rate F_s as low as can be accepted. Preemphsis can reduce $|\rho_i|$ and increase α_i. The low F_s implies only a few sampling data points. They showed that as F_s is increased, wordlength β must also be increased. Recall that the bound on quantization Δ is proportion to $2^{-\beta}$.

Another point we note is that, in practice, we require that linear prediction give a good result, that is, the prediction error α_i be small. However, smaller α_i means an ill-conditioned problem, and the computed result may be more sensitive to quantization error. Consequently, we can not obtain an arbitrarily small value of α_i and a compromise must be made.

3.2 Stability of Split Levinson Algorithm

Split Levinson algorithm(SLA) was introduced recently in order to increase the computation speed. It requires only $0.5n^2+O(n)$ arithmetic operations as it manipulates symmetric or anti-symmetric vector. In the following, we state some relevant results which will be used in this report. The details are given in [8]. We know that the Levinson algorithm recursively solves the system,

$$T_{n+1}\begin{pmatrix} 1 \\ \underline{a_n} \end{pmatrix} = (\alpha_n\ 0\ \cdots\ 0)^t \qquad (3.10)$$

and the recurrence is

$$\underline{a}_i = \begin{pmatrix} \underline{a}_{i-1} + \rho_i \hat{\underline{a}}_{i-1} \\ \rho_i \end{pmatrix} \tag{3.11}$$

Note that the vector $\underline{a}_i$ in this chapter is defined as

$$\underline{a}_i^t = (a_{i,1} \ \cdots \ a_{i,i})$$

Define a new vector

$$\underline{s}_{i-1} = \underline{a}_{i-1} + \hat{\underline{a}}_{i-1} \tag{3.12}$$

After some manipulation, we get the SLA as,

$$\underline{s}_i = \begin{pmatrix} \underline{s}_{i-1} \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \underline{s}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix} \tag{3.13}$$

$$s_1 = 2 - 2d_1 \tag{3.14}$$

$$\underline{s}_2 = \begin{pmatrix} 1 + s_1 - d_2 \\ 1 + s_1 - d_2 \end{pmatrix} \tag{3.15}$$

$$t_i = 1 + \sum_{m=1}^{i-1} c_m s_{i-1,m} + c_i \tag{3.16}$$

$$d_i = \frac{t_i}{t_{i-1}} \tag{3.17}$$

$$\rho_i = 1 - \frac{d_i}{1 + \rho_{i-1}} \tag{3.18}$$

with initial conditions $t_0 = 1$, $\rho_0 = 0$. Since $|\rho_i| \leq 1$, $0 \leq d_i \leq 4$. Observe that $\underline{s}_i$ is a symmetric vector. In fact, the SLA solves the linear system,

$$T_{n-1}\underline{s}_{n-1} = -(\underline{c}_{n-1} + \hat{\underline{c}}_{n-1}) \tag{3.19}$$

In this chapter, we study the numerical stability properties of the SLA. In this section, we state the results and discuss the condition under which the algorithm is well-conditioned. We give the various proofs in the following sections.

The residual for the algorithm is defined as

$$\underline{r}_n = T_n \underline{\alpha}_n \tag{3.20}$$

The bound on the residual for SLA is

$$\|\underline{r}_n\| \leq \Delta \left(\frac{2n^4}{3} + 3n^3 + \frac{13n^2}{3} \right) \prod_{m=1}^{n} (1 + |\rho_m|) \tag{3.21}$$

for the fixed-point arithmetic, and

$$\|\underline{r}_n\| \leq 16\Delta \left(n^3 + n^2 + n \right) \prod_{m=1}^{n} (1 + |\rho_m|) \tag{3.22}$$

for the floating-point arithmetic.

From these bounds, we see that, similar to the Levinson algorithm, the SLA is also weakly stable, if the order n is not large and $|\rho_i|$ is not close to one. The bounds obtained here are almost the same as those for Levinson algorithm obtained by Cybenko. The main difference between SLA and Levinson is the power of n in the bounds. Can we obtain a bound which has same power of n as Levinson algorithm? The answer is yes, but the bound is too loose and will not be pursued here. We will derive the bounds in (3.21) and (3.22) in the following sections. An important question arises at this stage —' why the power of n is different in the bounds for the Levinson algorthm and SLA'. The reason is that after computing the symmetric vector $\underline{s}_n$, we need to go back to prediction coefficients $\underline{a}_n$. This requires a division by $(1 - z^{-1})$, if we use polynomial notation . If we want to design an AR

filter using SLA (which we can do by using Levinson algorithm), the AR filter will have a terminal stage for division by $(1 - z^{-1})$. This stages corresponds to a *marginally stable IIR filter* of order one having a pole at $z = 1$. The divisor $(1 - z^{-1})$ plays a very important role in the SLA. It is the bridge linking the prediction coefficients and symmetric vector. Given the symmetric vector, the only way to get prediction coefficients is the bridge. However, since there are errors in the symmetric vector due to the finite wordlength of the computer, the polynomial formed by the computed vector could not be divided by $1 - z^{-1}$. The bridge is broken in a finite precision implementaion of the SLA. Does it mean that the SLA is not stable? We point out that for anti-symmetric form of the SLA, the same result can be obtained.

At this point, we can compare the Levinson algorithm and SLA. The Levinson algorithm can be used to implement an AR or ARMA filter, while the SLA can not be used. The Levinson algorithm gives directly results of all the subsystems during the intermediate computations, the SLA does not. The computation time of Levinson algorithm is $n^2+O(n)$, the time of SLA is $0.5n^2 + O(n)$. However, from the previous discussion, we know that the problem is ill-conditioned when n is large. This means that both the algorithms may be used only when n is small. In such a situation, SLA may not save computation time significantly. It appears that Levinson algorithm is safer than SLA.

3.3 Residual of Split Levinson Algorithm

It is well known that errors are produced by the computer due to finite wordlengths. The multiplication and division will produce errors in the fixed-point arithmetic, and all elementary operations will introdue errors in floating-point arithmetic. Therefore, in the following we discuss the two case seperately.

For fixed-point arithmetic, the model of error analysis is given by

$$fx(a \pm b) = a \pm b,$$

$$fx(ab) = ab + \xi,$$

$$fx(\frac{a}{b}) = \frac{a}{b} + \zeta,$$

where a and b are fixed-point numbers. ξ and ζ are errors and satisfy $|\xi|, |\zeta| \leq \Delta$, where Δ depends on the configuration of the computer. Usually $\Delta = C\beta^{-b}$, where β is base which computer uses, b is the wordlength, and C is a constant.

For floating-point arithmetic, the model is

$$fl(a + b) = (a + b)(1 + \xi),$$

$$fl(ab) = ab(1 + \zeta),$$

$$fl(\frac{a}{b}) = \frac{a}{b}(1 + \eta),$$

where ξ, ζ, η are errors and also satisfy $|\xi|$, $|\zeta|$, $|\eta| \leq \Delta$.

Now using these models, we study the effect of finite wordlength on the SLA. We denote the computed values by superscript c as

$$s^c_{i,j} = s_{i,j} + \alpha_{i,j} \tag{3.23}$$

$$d_i^c = d_i + \beta_i \tag{3.24}$$

$$t_i^c = t_i + \gamma_i \tag{3.25}$$

$$\rho_i^c = \rho_i + \mu_i \tag{3.26}$$

In the computer, the SLA becomes

$$\underline{s}_i^c = \begin{pmatrix} \underline{s}_{i-1}^c \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \underline{s}_{i-1}^c \end{pmatrix} - d_i^c \begin{pmatrix} 1 \\ \underline{s}_{i-2}^c \\ 1 \end{pmatrix} + \underline{\theta}_i \tag{3.27}$$

$$d_i^c = \frac{t_i^c}{t_{i-1}^c} + \eta_i \tag{3.28}$$

$$t_i^c = 1 + \underline{c}_{i-1}\underline{s}_{i-1}^c + c_i + \sigma_i \tag{3.29}$$

In the above expression, the errors $\underline{\theta}_i$, η_i, σ_i are local errors and $\alpha_{i,j}$, β_i, γ_i, μ_i are total errors. Note that we will consider only the first order error terms in our analysis; the second and higher order terms will be ignored.

Now, we derive the relationship between local and total error. Using (3.23), (3.25) and (3.29), we have

$$\begin{aligned} \gamma_i &= t_i^c - t_i \\ &= 1 + \underline{c}_{i-1}\underline{s}_{i-1}^c + c_i + \sigma_i - (1 + \underline{c}_{i-1}\underline{s}_{i-1} + c_i) \\ &= \underline{c}_{i-1}\underline{\alpha}_{i-1} + \sigma_i \end{aligned} \tag{3.30}$$

Taking the same procedure, we also have

$$\beta_i = d_i^c - d_i = \frac{t_i^c}{t_{i-1}^c} + \eta_i - \frac{t_i}{t_{i-1}} = \frac{\gamma_i - d_i\gamma_{i-1}}{t_{i-1}} + \eta_i \tag{3.31}$$

$$
\begin{aligned}
\underline{\alpha}_i &= \underline{s}_i^c - \underline{s}_i \\
&= \begin{pmatrix} \underline{\alpha}_{i-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{\alpha}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{\alpha}_{i-2} \\ 0 \end{pmatrix} \\
&- \beta_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix} + \underline{\theta}_i
\end{aligned} \tag{3.32}
$$

where $\alpha_1 = s_1^c - s_1 = -2\eta_1$ and $\alpha_2 = -2\eta_1 - \eta_2$. The residual is defined as

$$
r_n = T_n \underline{s}_n^c - T_n \underline{s}_n = T_n \underline{\alpha}_n \tag{3.33}
$$

Substituting (3.32) into (3.33), we have

$$
\begin{aligned}
r_i &= T_i \left[\begin{pmatrix} \underline{\alpha}_{i-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{\alpha}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{\alpha}_{i-2} \\ 0 \end{pmatrix} - \beta_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix} \right] \\
&+ T_i \underline{\theta}_i
\end{aligned} \tag{3.34}
$$

Since T_i can be partitioned as

$$
T_i = \begin{pmatrix} T_{i-1} & \hat{\underline{c}}_{i-1} \\ \hat{\underline{c}}_{i-1}^t & 1 \end{pmatrix} \tag{3.35}
$$

so

$$
T_i \begin{pmatrix} \underline{\alpha}_{i-1} \\ 0 \end{pmatrix} = \begin{pmatrix} \underline{r}_{i-1} \\ \underline{c}_{i-1}^t \underline{\alpha}_{i-1} \end{pmatrix} \tag{3.36}
$$

Similarly, partitioning T_i as

$$T_i = \begin{pmatrix} 1 & \underline{c}_{i-1}^t \\ \underline{c}_{i-1} & T_{i-1} \end{pmatrix} \tag{3.37}$$

we get

$$T_i \begin{pmatrix} 0 \\ \underline{\alpha}_{i-1} \end{pmatrix} = \begin{pmatrix} \underline{c}_{i-1}^t \underline{\alpha}_{i-1} \\ \underline{r}_{i-1} \end{pmatrix} \tag{3.38}$$

Also, partitioning T_i as

$$T_i = \begin{pmatrix} 1 & \underline{c}_{i-2}^t & c_{i-1} \\ \underline{c}_{i-2} & T_{i-2} & \underline{c}_{i-2} \\ c_{i-1} & \underline{c}_{i-2}^t & 1 \end{pmatrix} \tag{3.39}$$

We may write,

$$T_i \begin{pmatrix} 0 \\ \underline{\alpha}_{i-2} \\ 0 \end{pmatrix} = \begin{pmatrix} \underline{c}_{i-2}^t \underline{\alpha}_{i-2} \\ \underline{r}_{i-2} \\ \underline{c}_{i-2}^t \underline{\alpha}_{i-2} \end{pmatrix} \tag{3.40}$$

Substituting (3.36), (3.38) and (3.40) into (3.34) and noting that

$$T_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix} = \begin{pmatrix} 1 + \underline{c}_{i-2}^t \underline{s}_{i-2} + c_{i-1} \\ \underline{c}_{i-2} + T_{i-2} \underline{s}_{i-2} + \hat{\underline{c}}_{i-2} \\ c_{i-1} + \hat{\underline{c}}_{i-2}^t \underline{s}_{i-2} + 1 \end{pmatrix} = \begin{pmatrix} t_{i-1} \\ \underline{0} \\ t_{i-1} \end{pmatrix}, \tag{3.41}$$

we get

$$\underline{r}_i = \begin{pmatrix} \underline{r}_{i-1} \\ \underline{c}_{i-1}^t \underline{\alpha}_{i-1} \end{pmatrix} + \begin{pmatrix} \underline{c}_{i-1} \underline{\alpha}_{i-1} \\ \underline{r}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} \underline{c}_{i-2} \underline{\alpha}_{i-2} \\ \underline{r}_{i-2} \\ \underline{c}_{i-2}^t \underline{\alpha}_{i-2} \end{pmatrix}$$

$$- \beta_i \begin{pmatrix} t_{i-1} \\ \underline{0} \\ t_{i-1} \end{pmatrix} + T_i \underline{\theta}_i \tag{3.42}$$

Using (3.30) and (3.31), we have,

$$\underline{r}_i = \begin{pmatrix} \underline{r}_{i-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{r}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{r}_{i-2} \\ 0 \end{pmatrix} + T_i \underline{\theta}_i$$

$$+ \begin{pmatrix} -\sigma_i + d_i \sigma_{i-1} - \eta_i t_{i-1} \\ \underline{0} \\ -\sigma_i + d_i \sigma_{i-1} - \eta_i t_{i-1} \end{pmatrix} \tag{3.43}$$

For the discussion of the bound of the residual, we seperate (3.43) into

$$\underline{F}_{j,j} = \begin{pmatrix} -\sigma_j + d_j \sigma_{j-1} - \eta_j t_{j-1} \\ \underline{0} \\ -\sigma_j + d_j \sigma_{j-1} - \eta_j t_{j-1} \end{pmatrix} + T_j \underline{\theta}_j \tag{3.44}$$

$$\underline{F}_{j-1,j} = \underline{0} \tag{3.45}$$

and

$$\underline{F}_{i,j} = \begin{pmatrix} \underline{F}_{i-1,j} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{F}_{i-1,j} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{F}_{i-2,j} \\ 0 \end{pmatrix} \tag{3.46}$$

Now, we prove that

$$\underline{r}_i = \sum_{j=1}^{i} \underline{F}_{i,j} \tag{3.47}$$

Suppose that $\underline{r}_l = \sum_{j=1}^{l} \underline{F}_{l,j}$ is true when $l \leq i-1$. Then

$$\begin{aligned}
\sum_{j=1}^{i} \underline{F}_{i,j} &= \begin{pmatrix} \sum_{j=1}^{i-1} \underline{F}_{i-1,j} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \sum_{j=1}^{i-1} \underline{F}_{i-1,j} \end{pmatrix} \\
&- d_i \begin{pmatrix} 0 \\ \sum_{j=1}^{i-2} \underline{F}_{i-2,j} \\ 0 \end{pmatrix} + \underline{F}_{i,i} \\
&= \begin{pmatrix} \underline{r}_{i-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{r}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{r}_{i-2} \\ 0 \end{pmatrix} \\
&+ T_i \underline{\theta}_i + \begin{pmatrix} -\sigma_i + d_i \sigma_{i-1} - \eta_i t_{i-1} \\ \underline{0} \\ -\sigma_i + d_i \sigma_{i-1} - \eta_i t_{i-1} \end{pmatrix} \\
&= \underline{r}_i
\end{aligned} \tag{3.48}$$

Note that $r_1 = -2(\sigma_1 + \eta_1) + T_1 \underline{\theta}_1 = F_{1,1}$. Thus (3.47) is true by mathematical induction. Here, the vector $\underline{F}_{j,j}$ represents the error produced during the computation at jth stage, and $\underline{F}_{i,j}$ represents the error produced at jth stage propagated into $\underline{s}_i$.

3.4 The bound of residual

We have obtained the recurrence relation of the residual in the previous section. In this section, we derive a bound on the residual vector. We know

$$\underline{s}_{i-1} = \underline{a}_{i-1} + \hat{\underline{a}}_{i-1} \tag{3.49}$$

Since $\|\underline{a}_{i-1}\| \leq \prod_{m=1}^{i-1}(1 + |\rho_m|) - 1$ [6], have

$$\|\underline{s}_{i-1}\| \leq 2\left[\prod_{m=1}^{i-1}(1 + |\rho_m|) - 1\right] \tag{3.50}$$

We will use this relation in the following discussion. Before we get the bound of residual, we firstly prove two lemmas.

Lemma 5 *If a symmetric vector $\underline{v}_i$ satisfies the recurrence*

$$\underline{v}_i = \begin{pmatrix} \underline{v}_{i-1} \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ \underline{v}_{i-1} \end{pmatrix} - d_i \begin{pmatrix} 0 \\ \underline{v}_{i-2} \\ 0 \end{pmatrix} \tag{3.51}$$

$$d_i = (1 + \rho_i)(1 - \rho_{i-1}) \tag{3.52}$$

then another vector $\underline{A}_i$ can be defined as

$$\underline{A}_i = (1 + \rho_i) \begin{pmatrix} 0 \\ \underline{v}_{i-1} \end{pmatrix} - \underline{v}_i. \tag{3.53}$$

The vector A_i satisfies the recurrence,

$$\underline{A}_i = \begin{pmatrix} \underline{A}_{i-1} \\ 0 \end{pmatrix} - \rho_i \begin{pmatrix} 0 \\ \hat{\underline{A}}_{i-1} \end{pmatrix}. \tag{3.54}$$

Also,

$$\|\underline{v}_i\| \leq (2i-1)\|\underline{A}_i\| \tag{3.55}$$

Proof: Substituting (3.53) into both sides of (3.54), we obtaine (3.51). This prove that we can define $\underline{A}_i$ as in (3.53). Noting that

$$\hat{\underline{A}}_i = (1+\rho_i)\begin{pmatrix} \underline{v}_{i-1} \\ 0 \end{pmatrix} - \underline{v}_i \tag{3.56}$$

we have

$$\underline{A}_i = \begin{pmatrix} \underline{A}_{i-1} \\ 0 \end{pmatrix} - \rho_i \begin{pmatrix} 0 \\ \hat{\underline{A}}_{i-1} \end{pmatrix}. \tag{3.57}$$

From (3.53) and (3.57), we have

$$\underline{v}_i = (S_i - I_i)^{-1}(\underline{A}_i - \hat{\underline{A}}_i) - \hat{\underline{A}}_i \tag{3.58}$$

where

$$S_i = \begin{pmatrix} 0 & 0 & \cdots & 0 & 0 \\ 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \tag{3.59}$$

and I_i is the identity matrix of dimension i. The inverse of $S_i - I_i$ is the lower triangular matrix having all elements on and below the main diagonal equal to -1. So from (3.58), we have

$$\underline{v}_i = (S_i - I_i)^{-1}\underline{A}_i + L_i\hat{\underline{A}}_i, \tag{3.60}$$

where L_i is the lower triangular matrix having all elements on the main diagonal equal to 0 and all elements below the main diagonal equal to unity. The norm of $\underline{v}_i$ is bounded as,

$$\begin{aligned} \|\underline{v}_i\| &\leq \|(S_i - I_i)^{-1}\underline{A}_i\| + \|L_i\hat{\underline{A}}_i\| \\ &\leq i\|\underline{A}_i\| + (i-1)\|\underline{A}_i\| \\ &= (2i-1)\|\underline{A}_i\| \end{aligned} \tag{3.61}$$

At this point, we have proved the lemma. Now we state and prove Lemma 6.

Lemma 6 *If a vector $\underline{A}_i$ satisfies*

$$\underline{A}_{i+1} = \begin{pmatrix} \underline{A}_i \\ 0 \end{pmatrix} - \rho_{i+1} \begin{pmatrix} 0 \\ \hat{\underline{A}}_i \end{pmatrix} \tag{3.62}$$

and $\|\underline{A}_j\| \leq B$, then

$$\|\underline{A}_i\| \leq B \prod_{m=j+1}^{i} (1 + |\rho_m|) \tag{3.63}$$

Proof: Suppose that $\|\underline{A}_i\| \leq B\prod_{m=j+1}^{i}(1+|\rho_m|)$, then

$$\begin{aligned} \|\underline{A}_{i+1}\| &\leq \|\underline{A}_i\| + |\rho_{i+1}|\|\hat{\underline{A}}_i\| \\ &\leq (1 + |\rho_{i+1}|)\|\underline{A}_i\| \\ &\leq B \prod_{m=j+1}^{i+1} (1 + |\rho_m|) \end{aligned} \tag{3.64}$$

Since $\|\underline{A}_j\| \leq B$. By induction, we have proved the lemma.

Using the Lemmas 5 and 6 in (3.46), we get,

$$\|\underline{F}_{i,j}\| \leq (2i-1)B \prod_{m=j+1}^{i} (1 + |\rho_m|) \tag{3.65}$$

where B is determined by $\underline{F}_{j,j}$ as defined in (3.44). From (3.44) and (3.53), we have

$$\underline{A}_j = -\underline{F}_{j,j} \tag{3.66}$$

For fixed-point arithmetic, we have

$$|\sigma_i| \leq j\Delta \tag{3.67}$$

$$|\eta_j t_{j-1}| \leq \Delta|t_{j-1}| \leq 2\Delta \prod_{m=1}^{j-2}(1+|\rho_m|) \tag{3.68}$$

$$|\theta_{i,j}| \leq \Delta \tag{3.69}$$

and

$$\begin{aligned} \|\underline{F}_{j,j}\| &\leq 2|\sigma_j| + 2d_j|\sigma_{j-1}| + \|T_j\underline{\theta}_j\| + 2|\eta_j t_{j-1}| \\ &\leq \Delta(j^2+4)\prod_{m=1}^{j}(1+|\rho_m|) \\ &= B \end{aligned} \tag{3.70}$$

Substituting (3.70) into (3.65), we get

$$\|\underline{F}_{i,j}\| \leq (2i-1)\Delta(j^2+4)\prod_{m=1}^{i}(1+|\rho_m|) \tag{3.71}$$

and

$$\begin{aligned} \|\underline{r}_i\| &\leq \sum_{j=1}^{i}\|\underline{F}_{i,j}\| \\ &\leq \Delta\left(\frac{2i^4}{3}+\frac{2i^3}{3}+\frac{47i^2}{6}-\frac{25i}{6}\right)\prod_{m=1}^{i}(1+|\rho_m|) \end{aligned} \tag{3.72}$$

Writing the above equation for $i = n$, we obtain (3.21). For floating-point arithmetic, we have

$$|\sigma_j| \leq j\Delta(2+\|\underline{s}_{j-1}\|)$$

$$\leq 2j\Delta \prod_{m=1}^{j-1}(1+|\rho_m)| \tag{3.73}$$

$$|\eta_j t_{j-1}| \leq \Delta|t_{j-1}|$$

$$\leq 2\Delta \prod_{m=1}^{j-2}(1+|\rho_m|) \tag{3.74}$$

and since

$$fl(\underline{s}_i) = fl\left\{ fl\left[\begin{pmatrix} \underline{s}_{i-1} \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \underline{s}_{i-1} \end{pmatrix}\right] - fl\left[d_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix}\right]\right\}$$

$$= (1+\zeta)\left\{(1+\xi)\left[\begin{pmatrix} \underline{s}_{i-1} \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \underline{s}_{i-1} \end{pmatrix}\right]\right\}$$

$$- (1+\zeta)(1+\eta)d_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix}$$

so

$$\underline{\theta}_i = (\zeta+\xi)\left[\begin{pmatrix} \underline{s}_{i-1} \\ 1 \end{pmatrix} + \begin{pmatrix} 1 \\ \underline{s}_{i-1} \end{pmatrix}\right] - (\zeta+\eta)d_i \begin{pmatrix} 1 \\ \underline{s}_{i-2} \\ 1 \end{pmatrix} \tag{3.75}$$

where ζ, ξ, η are local errors and satisfy $|\zeta|$, $|\xi|$, $|\eta| \leq \Delta$. The bound on $\underline{\theta}_i$ can be written as

$$\|\underline{\theta}_j\| \leq 2\Delta\left[2 + 2\|\underline{s}_{j-1}\| + 2d_j + d_j\|\underline{s}_{j-2}\|\right]$$

$$\leq 12\Delta \prod_{m=1}^{j}(1+|\rho_m|) \tag{3.76}$$

so

$$\begin{aligned} \|\underline{F}_{j,j}\| &\leq 2|\sigma_j| + 2d_j|\sigma_{j-1}| + \|T_j\underline{\theta}_j\| + 2|\eta_j t_{j-1}| \\ &\leq \Delta(20j+4)\prod_{m=1}^{j}(1+|\rho_m|) = B \end{aligned}$$

from (3.65), we get

$$\|\underline{F}_{i,j}\| \leq (2i-1)\Delta(20j+4)\prod_{m=1}^{i}(1+|\rho_m|) \tag{3.77}$$

and

$$\|\underline{r}_i\| \leq (20i^3+18i^2-14i)\Delta\prod_{m=1}^{i}(1+|\rho_m|) \tag{3.78}$$

Writing the above equation for $i=n$, we obtain (3.22).

From these bounds we see that if $|\rho_i|$ is not close to one and n is not large, the residual is small. This means that the solution of Toeplitz system is accurate. If n is large, and the problem is ill-conditioned, the residual may be large.

It is worthwhile to mention here that we cannot define $\underline{A}_i$ by $\underline{v}_i = \underline{A}_i + \hat{\underline{A}}_i$ in lemma 5 as this definition will require a polynomial devision by $(1-z^{-1})$ to get $\underline{A}_n$ from $\underline{v}_n$. However, it is not guaranteed that this requirement will be satisfied due to errors. In filter design, this means that a pole exists on the unit circle and the corresponding filter is unstable. This could mean that the SLA is not strictly stable. Consequently, we have to adopt an alternative route to reach our objective.

Chapter 4

The Fast and Superfast Algorithms for Block Toeplitz Matries

In the last two chapters, we have studied the fast algorithms to solving Toeplitz systems. The algorithms have the complexity of computation with order $O(n^2)$. They are more efficient than the classical methods. However, this is not the end of searching for more efficient algorithms. Another category, superfast algorithm, that is , algorithms with order $O(n \log^2 n)$ complexity, will be studied in this chapter.

Until now, the Toeplitz system that has been studied corresponds to the scalar case, that is, the elements of Toeplitz matrix are scalars. The block Toeplitz system, that is, each element of the Toeplitz matrix is also a submatrix, appears in applications such as image processing. The fast and superfast algorithm for the block case is also studied in this chapter.

We firstly study the fast algorithm for block Toeplitz system. We proved the Gohberg-

Semencul formula for a general Toeplitz matrix,and then, introduce the Schur algorithm for the block matrix. The superfast algorithm will be studied.

4.1 The Fast Algorithm for Block Toeplitz Matrices

The Levinson algorithm for solving the scalar Toeplitz systems has been studied in the previous chapters. The details of algorithms for inverting a block Toeplitz matrix are studied in this section [16, 17, 18]. The results for the block Toeplitz matrix are very similar to those for scalar case.

Consider a Toeplitz matrix

$$T_{p,n} = (T_{i-j}) \quad i,j = 1,2,\cdots,n \tag{4.1}$$

where each T_i is $p \times p$ matrix. We want to get a fast algorithm which gives an inversion of $T_{p,n}$.

We can partition $T_{p,n}$ into

$$T_{p,n} = \begin{bmatrix} T_{p,n-1} & \hat{\underline{T}}_{-n} \\ \hat{\underline{T}}_{n}^{t} & T_0 \end{bmatrix} \tag{4.2}$$

where

$$\hat{\underline{T}}_{-n} = \begin{bmatrix} T_{-n} \\ \vdots \\ T_{-1} \end{bmatrix}$$

$$\hat{\underline{T}}_{n}^{t} = \begin{bmatrix} T_n \cdots T_1 \end{bmatrix}$$

In this chapter, superscript t represents block transpose, and ' represents the common transpose.

For a matrix

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

We know the inverse of A is

$$A^{-1} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$$

where

$$\begin{aligned} a_{11} &= A_{11}^{-1} + A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} \\ a_{12} &= -A_{11}^{-1}A_{12}(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \\ a_{21} &= -(A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1}A_{21}A_{11}^{-1} \\ a_{22} &= (A_{22} - A_{21}A_{11}^{-1}A_{12})^{-1} \end{aligned}$$

So if we use this formula for $T_{p,n}$, we have

$$T_{p,n}^{-1} = \begin{bmatrix} T_{p,n-1}^{-1} + \hat{\underline{W}}_n \alpha_n^{-1} \hat{\underline{V}}_n^t & \hat{\underline{W}}_n \alpha_n^{-1} \\ \alpha_n^{-1} \hat{\underline{V}}_n^t & \alpha_n^{-1} \end{bmatrix} \tag{4.3}$$

where

$$\hat{\underline{W}}_n = -T_{p,n-1}^{-1} \hat{\underline{T}}_{-n} \tag{4.4}$$

$$\hat{\underline{V}}_n^t = -\hat{\underline{T}}_n^t T_{p,n-1}^{-1} \tag{4.5}$$

$$\alpha_n = T_0 - \hat{\underline{T}}_n^t T_{p,n-1}^{-1} \hat{\underline{T}}_{-n} \tag{4.6}$$

From (4.4), we get

$$\underline{W}_{n+1} = -T_{p,n}^{-t}\underline{T}_{-(n+1)} \tag{4.7}$$

If we partition $T_{p,n}^{t}$, we get

$$T_{p,n}^{t} = \begin{bmatrix} T_{p,n-1}^{t} & \underline{\hat{T}}_{n} \\ \underline{\hat{T}}_{-n}^{t} & T_0 \end{bmatrix} \tag{4.8}$$

and inverse of $T_{p,n}^{t}$ is

$$T_{p,n}^{-t} = \begin{bmatrix} T_{p,n-1}^{-t} + \underline{\hat{X}}_n \beta_n^{-1} \underline{\hat{Z}}_n^{t} & \underline{\hat{X}}_n \beta_n^{-1} \\ \beta_n^{-1} \underline{\hat{Z}}_n^{t} & \beta_n^{-1} \end{bmatrix} \tag{4.9}$$

where

$$\underline{\hat{X}}_n = -T_{p,n-1}^{-t}\underline{\hat{T}}_n \tag{4.10}$$

$$\underline{\hat{Z}}_n^{t} = -\underline{\hat{T}}_{-n}^{t} T_{p,n-1}^{-t} \tag{4.11}$$

$$\beta_n = T_0 - \underline{\hat{T}}_{-n}^{t} T_{p,n-1}^{-t} \underline{\hat{T}}_n \tag{4.12}$$

Substituting (4.9) into (4.7), we have

$$\begin{aligned} \underline{W}_{n+1} &= -\begin{bmatrix} T_{p,n-1}^{-t} + \underline{\hat{X}}_n \beta_n^{-1} \underline{\hat{Z}}_n^{t} & \underline{\hat{X}}_n \beta_n^{-1} \\ \beta_n^{-1} \underline{\hat{Z}}_n^{t} & \beta_n^{-1} \end{bmatrix} \begin{bmatrix} \underline{T}_{-n} \\ T_{-(n+1)} \end{bmatrix} \\ &= -\begin{bmatrix} T_{p,n-1}^{-t}\underline{T}_{-n} \\ 0 \end{bmatrix} - \begin{bmatrix} \underline{\hat{X}}_n \\ I_p \end{bmatrix} \beta_n^{-1} u_n \end{aligned} \tag{4.13}$$

where

$$u_n = \underline{\hat{Z}}_n^{t}\underline{T}_{-n} + T_{-(n+1)} \tag{4.14}$$

In this chapter, we define

$$J_{p,n} = \begin{bmatrix} 0 & \cdots & 0 & I_p \\ 0 & \cdots & I_p & 0 \\ \vdots & \vdots & \vdots & \vdots \\ I_p & \cdots & 0 & 0 \end{bmatrix} \tag{4.15}$$

Note that $J_{p,n}J_{p,n} = I_{p,n}$.

From (4.7), we finally have

$$\underline{W}_{n+1} = \begin{bmatrix} \underline{W}_n \\ 0 \end{bmatrix} - \begin{bmatrix} \hat{\underline{X}}_n \\ I_p \end{bmatrix} \beta_n^{-1} u_n \tag{4.16}$$

From (4.5), we have

$$\hat{\underline{V}}_{n+1}^t = -\hat{\underline{T}}_{n+1}^t J_{p,n} J_{p,n} T_{p,n-1}^{-t}$$

So

$$\begin{aligned} \underline{V}_{n+1}^t &= -\underline{T}_{n+1}^t T_{p,n-1}^{-t} \\ &= \begin{bmatrix} \underline{V}_n^t & 0 \end{bmatrix} - o_n \beta_n^{-1} \begin{bmatrix} \hat{\underline{Z}}_n^t & I \end{bmatrix} \end{aligned} \tag{4.17}$$

where

$$o_n = \underline{T}_n^t \hat{\underline{X}}_n + T_{n+1} = \underline{T}_{n+1}^t \begin{bmatrix} \hat{\underline{X}}_n \\ I_p \end{bmatrix} \tag{4.18}$$

Taking the same procedure on $\underline{X}_{n+1}$ and $\underline{Z}_{n+1}^t$, we have

$$\underline{X}_{n+1} = \begin{bmatrix} \underline{X}_n \\ 0 \end{bmatrix} - \begin{bmatrix} \hat{\underline{W}}_n \\ I_p \end{bmatrix} \alpha_n^{-1} e_n \tag{4.19}$$

$$\underline{Z}_{n+1}^t = \begin{bmatrix} \underline{Z}_n^t & 0 \end{bmatrix} - f_n \alpha_n^{-1} \begin{bmatrix} \hat{\underline{V}}_n^t & I_p \end{bmatrix} \tag{4.20}$$

where

$$e_n = \underline{\hat{V}}_n^t \underline{T}_n + T_{n+1} \tag{4.21}$$

$$f_n = \underline{T}_{-n}^t \underline{\hat{W}}_n + T_{-(n+1)} \tag{4.22}$$

Note that the u_n, o_n, e_n, f_n, α_n and β_n are $p \times p$ matrices. Now we prove some of relations between u_n, o_n,e_n,f_n,α_n and β_n.

Using (4.4) and (4.11), we have

$$u_n - f_n = \underline{\hat{Z}}_n^t \underline{T}_{-n} - \underline{T}_{-n}^t \underline{\hat{W}}_n = 0$$

or

$$u_n = f_n \tag{4.23}$$

By the same steps, we can prove

$$o_n = e_n \tag{4.24}$$

Let us find the relation of α_n and β_n now. From (4.6), we have

$$\alpha_{n+1} - \alpha_n = -o_n \beta_n^{-1} u_n$$

or

$$\alpha_{n+1} = \alpha_n - o_n \beta_n^{-1} u_n \tag{4.25}$$

similarly,

$$\beta_{n+1} = \beta_n - f_n \alpha_n^{-1} e_n \tag{4.26}$$

From (4.9), we have

$$T_{p,n}^{-1} = J_{p,n} T_{p,n}^{-t} J_{p,n}$$

$$= \begin{bmatrix} \beta^{-1} & \beta^{-1}\underline{Z}_n^t \\ \underline{X}_n\beta_n^{-1} & T_{p,n-1}^{-1} + \underline{X}_n\beta_n^{-1}\underline{Z}_n^t \end{bmatrix} \tag{4.27}$$

we observe that

$$(T_{p,n}^{-1})_{0,j} = \beta_n^{-1}\begin{bmatrix} I_p & \underline{Z}_n^t \end{bmatrix} \tag{4.28}$$

$$(T_{p,n}^{-1})_{i,0} = \begin{bmatrix} I_p \\ \underline{X}_n \end{bmatrix}\beta_n^{-1} \tag{4.29}$$

$$(T_{p,n}^{-1})_{i+1,j+1} = (T_{p,n-1}^{-1} + \underline{X}_n\beta_n^{-1}\underline{Z}_n^t)_{i,j} \tag{4.30}$$

From (4.3), we also have

$$(T_{p,n}^{-1})_{i,j} = (T_{p,n-1}^{-1} + \underline{\hat{W}}_n\alpha_n^{-1}\underline{\hat{V}}_n^t)_{i,j} \tag{4.31}$$

Combining (4.30) and (4.31), we get

$$(T_{p,n}^{-1})_{i+1,j+1} = (T_{p,n}^{-1})_{i,j} + (\underline{X}_n\beta_n^{-1}\underline{Z}_n^t - \underline{\hat{W}}_n\alpha_n^{-1}\underline{\hat{V}}_n^t)_{i,j} \tag{4.32}$$

The equation (4.32) can also be written as

$$\begin{aligned} T_{p,n}^{-1} = & \begin{bmatrix} I_p & 0 & \cdots & 0 \\ X_1 & I_p & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ X_n & \cdots & X_1 & I_p \end{bmatrix}\begin{bmatrix} \beta_n^{-1} & 0 & \cdots & 0 \\ 0 & \beta_n^{-1} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_n^{-1} \end{bmatrix}\begin{bmatrix} I_p & Z_1 & \cdots & Z_n \\ 0 & I_p & \cdots & Z_{n-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & I_p \end{bmatrix} \\ - & \begin{bmatrix} 0 & 0 & \cdots & 0 \\ W_n & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ W_1 & \cdots & W_n & 0 \end{bmatrix}\begin{bmatrix} \alpha_n^{-1} & 0 & \cdots & 0 \\ 0 & \alpha_n^{-1} & 0 & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha_n^{-1} \end{bmatrix}\begin{bmatrix} 0 & V_n & \cdots & V_1 \\ 0 & 0 & \cdots & V_2 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix} \end{aligned} \tag{4.33}$$

This is a very important result.

From (4.4) and (4.5), we have

$$T^t_{p,n-1}\underline{W}_n + \underline{T}_{-n} = 0 \tag{4.34}$$

$$\underline{V}^t_n T^t_{p,n-1} + \underline{T}^t_n = 0 \tag{4.35}$$

Now

$$T^t_{p,n}\begin{bmatrix} I_p \\ \underline{W}_n \end{bmatrix} = \begin{bmatrix} T_0 & \underline{T}^t_n \\ \underline{T}_{-n} & T^t_{p,n-1} \end{bmatrix}\begin{bmatrix} I_p \\ \underline{W}_n \end{bmatrix}$$

$$= \begin{bmatrix} \alpha_n \\ 0 \end{bmatrix} \tag{4.36}$$

Similarly,

$$\begin{bmatrix} I_p & \underline{V}^t_n \end{bmatrix} T^t_{p,n} = \begin{bmatrix} \alpha_n & 0 \end{bmatrix} \tag{4.37}$$

$$\begin{bmatrix} I_p & \underline{Z}^t_n \end{bmatrix} T_{p,n} = \begin{bmatrix} \beta_n & 0 \end{bmatrix} \tag{4.38}$$

$$T_{p,n}\begin{bmatrix} I_p \\ \underline{X}_n \end{bmatrix} = \begin{bmatrix} \beta_n \\ 0 \end{bmatrix} \tag{4.39}$$

Now let

$$\underline{Z}^t_n \leftarrow \begin{bmatrix} I_p & \underline{Z}^t_n \end{bmatrix}$$

$$\underline{X}_n \leftarrow \begin{bmatrix} I_p \\ \underline{X}_n \end{bmatrix}$$

$$\underline{V}^t_n \leftarrow \begin{bmatrix} I_p & \underline{V}^t_n \end{bmatrix}$$

$$\underline{W}_n \leftarrow \begin{bmatrix} I_p \\ \underline{W}_n \end{bmatrix}$$

From previous discussion, we have

$$T_{p,n}^t \underline{W}_n = \begin{bmatrix} \alpha_n \\ 0 \end{bmatrix} \tag{4.40}$$

$$\underline{V}_n^t T_{p,n}^t = \begin{bmatrix} \alpha_n & 0 \end{bmatrix} \tag{4.41}$$

$$T_{p,n} \underline{X}_n = \begin{bmatrix} \beta_n \\ 0 \end{bmatrix} \tag{4.42}$$

$$\underline{Z}_n^t T_{p,n} = \begin{bmatrix} \beta_n & 0 \end{bmatrix} \tag{4.43}$$

$$\underline{W}_{n+1} = \begin{bmatrix} \underline{W}_n \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \hat{\underline{X}}_n \end{bmatrix} \beta_n^{-1} u_n \tag{4.44}$$

$$\underline{V}_{n+1}^t = \begin{bmatrix} \underline{V}_n & 0 \end{bmatrix} - o_n \beta_n^{-1} \begin{bmatrix} 0 & \hat{\underline{Z}}_n^t \end{bmatrix} \tag{4.45}$$

$$\underline{X}_{n+1} = \begin{bmatrix} \underline{X}_n \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \hat{\underline{W}}_n \end{bmatrix} \alpha_n^{-1} e_n \tag{4.46}$$

$$\underline{Z}_{n+1}^t = \begin{bmatrix} \underline{Z}_n^t & 0 \end{bmatrix} - f_n \alpha_n^{-1} \begin{bmatrix} 0 & \hat{\underline{V}}_n^t \end{bmatrix} \tag{4.47}$$

or in polynomial form

$$W_{n+1}(z) = W_n(z) - z\hat{X}_n(z)\beta_n^{-1} u_n \tag{4.48}$$

$$V_{n+1}^t(z) = V_n^t(z) - z o_n \beta_n^{-1} \hat{Z}_n^t(z) \tag{4.49}$$

$$X_{n+1}(z) = X_n(z) - z\hat{W}_n(z)\alpha_n^{-1} e_n \tag{4.50}$$

$$Z_{n+1}^t(z) = Z_n^t(z) - z f_n \alpha_n^{-1} \hat{V}_n^t(z) \tag{4.51}$$

where $W_0 = V_0 = X_0 = Z_0 = I_p$ and $\alpha_0 = \beta_0 = T_0$. We can see that (4.48)–(4.51) have the same recursive form as the Levinson algorithm.

If we set the $p = 1$, that is, $T_{p,n}$ is a scalar matrix, then from (4.6) and (4.12), and noting that β_n and α_n are scalars, we get

$$\beta_n = \alpha_n$$

From (4.40) – (4.43), we have

$$\underline{X}_n = \underline{V}_n \quad and \quad \underline{W}_n = \underline{Z}_n$$

It is clear that these are two unknown vectors in this case and the algorithm can be rewritten as

$$W_{n+1}(z) = W_n(z) - z\omega_{n+1}\hat{X}_n(z) \tag{4.52}$$

$$X_{n+1}(z) = X_n(z) - z\gamma_{n+1}\hat{W}_n(z) \tag{4.53}$$

where

$$\omega_{n+1} = \beta_n^{-1}u_n = \alpha_n^{-1}f_n$$

$$\gamma_{n+1} = o_n\beta_n^{-1} = \alpha_n^{-1}e_n$$

This is the generalized Levinson algorithm for a scalar Toeplitz matrix which is non-symmetric.

If we assume that $T_{p,n}$ is symmetric, then we also have

$$\underline{W}_n = \underline{V}_n = \underline{X}_n = \underline{Z}_n$$

and

$$\omega_{n+1} = \gamma_{n+1}$$

The algorithm becomes the classical Levinson algorithm which was studied in chapter 2.

4.2 The Schur Algorithm for Block Toeplitz Matrices

In this section, we use the transformation on the result of last section to get Schur algorithm on block case.

Let

$$\begin{bmatrix} \underline{P}_{n-k} & \underline{Q}_{n-k} \end{bmatrix} = \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} T_{p,n} \begin{bmatrix} \underline{X}_{k-1} & 0 \\ 0 & \underline{\hat{W}}_{k-1} \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix} \tag{4.54}$$

Then

$$\begin{aligned} \underline{P}_{n-k} &= \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} T_{p,n} \begin{bmatrix} \underline{X}_{k-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} \begin{bmatrix} T_{p,k-1} & * \\ L_1 & * \end{bmatrix} \begin{bmatrix} \underline{X}_{k-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \\ &= L_1 \underline{X}_{k-1} \end{aligned} \tag{4.55}$$

The first row of L_1 is $\underline{\hat{T}}_k^t$ so

$$P_{n-k,0} = \underline{\hat{T}}_k^t \underline{X}_{k-1} = o_{k-1} = e_{k-1} \tag{4.56}$$

Note the star * represents the term which we do not care about. For $\underline{Q}_{n-k}$ we have

$$\begin{aligned}
\underline{Q}_{n-k} &= \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} T_{p,n} \begin{bmatrix} 0 \\ \underline{\hat{W}}_{k-1} \\ 0 \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} \begin{bmatrix} T_{p,k} & * \\ L_2 & * \end{bmatrix} \begin{bmatrix} 0 \\ \underline{\hat{W}}_{k-1} \\ 0 \\ \vdots \end{bmatrix} \\
&= \begin{bmatrix} \alpha_{k-1} \\ * \end{bmatrix}
\end{aligned} \tag{4.57}$$

so $Q_{n-k,0} = \alpha_{k-1}$.

Now we are going to obtain the recurrence relation of $\underline{P}_{n-k}$ and $\underline{Q}_{n-k}$.

$$\underline{P}_{n-k-1} = \begin{bmatrix} O_{k+1} & I_{n-k} \end{bmatrix} T_{p,n} \begin{bmatrix} \underline{X}_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$$= \begin{bmatrix} O_{k+1} & I_{n-k} \end{bmatrix} T_{p,n} \left\{ \begin{bmatrix} \underline{X}_{k-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \underline{\hat{W}}_{k-1} \\ 0 \\ \vdots \end{bmatrix} \alpha_{k-1}^{-1} e_{k-1} \right\}$$

$$= \begin{bmatrix} P_{n-k,1} \\ \vdots \\ P_{n-k,n-k} \end{bmatrix} - \begin{bmatrix} Q_{n-k,1} \\ \vdots \\ Q_{n-k,n-k} \end{bmatrix} \alpha_{k-1}^{-1} e_{k-1} \qquad (4.58)$$

Another recurrence relation of $\underline{Q}_{n-k}$ is

$$\underline{Q}_{n-k-1} = \begin{bmatrix} O_{k+1} & I_{n-k} \end{bmatrix} T_{p,n} \begin{bmatrix} 0 \\ \underline{\hat{W}}_k \\ 0 \\ \vdots \end{bmatrix}$$

$$= \begin{bmatrix} O_{k+1} & I_{n-k} \end{bmatrix} T_{p,n} \left\{ \begin{bmatrix} 0 \\ 0 \\ \underline{\hat{W}}_{k-1} \\ 0 \\ \vdots \end{bmatrix} - \begin{bmatrix} 0 \\ \underline{X}_{k-1} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \beta_{k-1}^{-1} u_{k-1} \right\}$$

$$= \begin{bmatrix} Q_{n-k,0} \\ \vdots \\ Q_{n-k,n-k-1} \end{bmatrix} - \begin{bmatrix} P_{n-k,0} \\ \vdots \\ P_{n-k,n-k-1} \end{bmatrix} \beta_{k-1}^{-1} u_{k-1} \qquad (4.59)$$

The initial condition is

$$\begin{bmatrix} \underline{P}_{n-1} & \underline{Q}_{n-1} \end{bmatrix} = \begin{bmatrix} T_1 & T_0 \\ \vdots & \vdots \\ T_n & T_{n-1} \end{bmatrix} \tag{4.60}$$

Now we define

$$\begin{bmatrix} \underline{R}_{n-k} & \underline{S}_{n-k} \end{bmatrix} = \begin{bmatrix} O_k & I_{n-k+1} \end{bmatrix} T^t_{p,n} \begin{bmatrix} \underline{W}_{k-1} & 0 \\ 0 & \underline{\hat{X}}_{k-1} \\ 0 & 0 \\ \vdots & \vdots \end{bmatrix} \tag{4.61}$$

Taking the same procedure, we can prove that

$$\underline{R}_{n-k-1} = \begin{bmatrix} R_{n-k,1} \\ \vdots \\ R_{n-k,n-k} \end{bmatrix} - \begin{bmatrix} S_{n-k,1} \\ \vdots \\ S_{n-k,n-k} \end{bmatrix} \beta^{-1}_{k-1} u_{k-1} \tag{4.62}$$

$$\underline{S}_{n-k-1} = \begin{bmatrix} S_{n-k,0} \\ \vdots \\ S_{n-k,n-k-1} \end{bmatrix} - \begin{bmatrix} R_{n-k,0} \\ \vdots \\ R_{n-k,n-k-1} \end{bmatrix} \alpha^{-1}_{k-1} e_{k-1} \tag{4.63}$$

$$R_{n-k,0} = f_{k-1}$$

$$S_{n-k,0} = \beta_{k-1}$$

The initial condition is

$$\begin{bmatrix} \underline{R}_{n-1} & \underline{S}_{n-1} \end{bmatrix} = \begin{bmatrix} T_{-1} & T_0 \\ \vdots & \vdots \\ T_{-n} & T_{-n+1} \end{bmatrix}$$

If we write the equations obtained in this section in polynomial form, we have

$$zP_{n-k-1}(z) = P_{n-k}(z) - Q_{n-k}(z)\alpha_{k-1}^{-1}e_{k-1} \quad (4.64)$$

$$Q_{n-k-1}(z) = -P_{n-k}(z)\beta_{k-1}^{-1}u_{k-1} + Q_{n-k}(z) \mod z^{n-k} \quad (4.65)$$

$$zR_{n-k-1}(z) = R_{n-k}(z) - S_{n-k}(z)\beta_{k-1}^{-1}u_{k-1} \quad (4.66)$$

$$S_{n-k-1}(z) = -R_{n-k}(z)\alpha_{k-1}^{-1}e_{k-1} + S_{n-k}(z) \mod z^{n-k} \quad (4.67)$$

where

$$P_{n-k,0} = e_{k-1}, \quad Q_{n-k,0} = \alpha_{k-1} \quad (4.68)$$

$$S_{n-k,0} = \beta_{k-1}, \quad R_{n-k,0} = u_{k-1} \quad (4.69)$$

4.3 The Superfast Algorithm for Block Toeplitz Matrices

In the previous two sections, we have discussed the Levinson and Schur algorithm for block Toeplitz matrix. Since their recursive structure is same as Levinson algorithm, the complexity of these algorithms are $O(n^2p^3)$. We study the fast Fourier transform (FFT) based superfast algorithm in this section.

Let us define

$$\Omega_{k+1} = \beta_k^{-1}u_k \quad (4.70)$$

$$\Gamma_{k+1} = \alpha_k^{-1}e_k \quad (4.71)$$

$$\Lambda_{k+1} = o_k\beta_k^{-1} \quad (4.72)$$

$$\Delta_{k+1} = f_k \alpha_k^{-1} \tag{4.73}$$

so we can write (4.48)–(4.51) as

$$\begin{bmatrix} X_k(z) & \hat{W}_k(z) \end{bmatrix} = \begin{bmatrix} X_{k-1}(z) & \hat{W}_{k-1}(z) \end{bmatrix} \begin{bmatrix} 1 & -\Omega_k \\ -z\Gamma_k & z \end{bmatrix} \tag{4.74}$$

$$\begin{bmatrix} Z_k^t(z) \\ \hat{V}_k^t(z) \end{bmatrix} = \begin{bmatrix} 1 & -\Delta_k z \\ -\Lambda_k & z \end{bmatrix} \begin{bmatrix} Z_{k-1}^t(z) \\ \hat{V}_{k-1}^t(z) \end{bmatrix} \tag{4.75}$$

and (4.64)–(4.67) as

$$\begin{bmatrix} P_{n-k-1}(z) & Q_{n-k-1}(z) \end{bmatrix} = \begin{bmatrix} P_{n-k}(z) & Q_{n-k}(z) \end{bmatrix} \begin{bmatrix} \frac{1}{z} & -\Omega_k \\ -\frac{1}{z}\Gamma_k & 1 \end{bmatrix} \tag{4.76}$$

$$\begin{bmatrix} R_{n-k-1}(z) & S_{n-k-1}(z) \end{bmatrix} = \begin{bmatrix} R_{n-k}(z) & S_{n-k}(z) \end{bmatrix} \begin{bmatrix} \frac{1}{z} & -\Gamma_k \\ -\frac{1}{z}\Omega_k & 1 \end{bmatrix} \tag{4.77}$$

Now let

$$\begin{bmatrix} A_k(z) & B_k(z) \\ zC_k(z) & zD_k(z) \end{bmatrix} = \prod_{i=1}^{k} \begin{bmatrix} 1 & -\Omega_i \\ -z\Gamma_i & z \end{bmatrix} \tag{4.78}$$

so

$$\begin{bmatrix} X_k(z) & \hat{W}_k(z) \end{bmatrix} = \begin{bmatrix} I_p & I_p \end{bmatrix} \begin{bmatrix} A_k(z) & B_k(z) \\ zC_k(z) & zD_k(z) \end{bmatrix} \tag{4.79}$$

We also can write (4.76) and (4.77) as

$$\begin{bmatrix} P_{n-k-1}(z) & Q_{n-k-1}(z) \end{bmatrix} = \begin{bmatrix} P_{n-1}(z) & Q_{n-1}(z) \end{bmatrix} \prod_{i=1}^{k} \begin{bmatrix} \frac{1}{z} & -\Omega_i \\ -\frac{1}{z}\Gamma_i & 1 \end{bmatrix} \tag{4.80}$$

$$\begin{bmatrix} R_{n-k-1}(z) & S_{n-k-1}(z) \end{bmatrix} = \begin{bmatrix} R_{n-1}(z) & S_{n-1}(z) \end{bmatrix} \prod_{i=1}^{k} \begin{bmatrix} \frac{1}{z} & -\Gamma_i \\ -\frac{1}{z}\Omega_i & 1 \end{bmatrix} \tag{4.81}$$

Now we prove the relation

$$\begin{bmatrix} z^{-k}A_k(z) & z^{1-k}B_k(z) \\ z^{-k}C_k(z) & z^{1-k}D_k(z) \end{bmatrix} = \prod_{i=1}^{k} \begin{bmatrix} \frac{1}{z} & -\Omega_i \\ -\frac{1}{z}\Gamma_i & 1 \end{bmatrix} \tag{4.82}$$

Proof: For k=1, (4.82) is true. Suppose that (4.82) is true for $k = l$, then for $k = l + 1$

$$\begin{aligned} \prod_{i=1}^{l+1} \begin{bmatrix} \frac{1}{z} & -\Omega_i \\ -\frac{1}{z}\Gamma_i & 1 \end{bmatrix} &= \prod_{i=1}^{l} \begin{bmatrix} \frac{1}{z} & -\Omega_i \\ -\frac{1}{z}\Gamma_i & 1 \end{bmatrix} \begin{bmatrix} \frac{1}{z} & -\Omega_{l+1} \\ -\frac{1}{z}\Gamma_{l+1} & 1 \end{bmatrix} \\ &= \begin{bmatrix} z^{-l}A_l(z) & z^{1-l}B_l(z) \\ z^{-l}C_l(z) & z^{1-l}D_l(z) \end{bmatrix} \begin{bmatrix} \frac{1}{z} & -\Omega_{l+1} \\ -\frac{1}{z}\Gamma_{l+1} & 1 \end{bmatrix} \\ &= \begin{bmatrix} z^{-l-1}\left(A_l(z) - zB_l(z)\Gamma_{l+1}\right) & z^{-l}\left(-A_l(z)\Omega_{l+1} + xB_l(z)\right) \\ z^{-l-1}\left(C_l(z) - zD_l(z)\Gamma_{l+1}\right) & z^{-l}\left(-C_l(z)\Omega_{l+1} + zD_l(z)\right) \end{bmatrix} \end{aligned}$$

Using (4.78), we can prove

$$\prod_{i=1}^{l+1} \begin{bmatrix} \frac{1}{z} & -\Omega_i \\ -\frac{1}{z}\Gamma_i & 1 \end{bmatrix} = \begin{bmatrix} z^{-(l+1)}A_{l+1}(z) & z^{1-(l+1)}B_{l+1}(z) \\ z^{-(l+1)}C_{l+1}(z) & z^{1-(l+1)}D_{l+1}(z) \end{bmatrix} \tag{4.83}$$

Thus (4.82) is proved by induction.

Using the same procedure we can prove

$$\prod_{i=1}^{k} \begin{bmatrix} \frac{1}{z} & \Gamma_i \\ -\frac{\Omega_i}{z} & 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{z}D_k(\frac{1}{z}) & C_k(\frac{1}{z}) \\ \frac{1}{z}B_k(\frac{1}{z}) & A_k(\frac{1}{z}) \end{bmatrix} \tag{4.84}$$

Now, we let

$$\begin{bmatrix} E_k(z) & zF_k(z) \\ G_k(z) & zH_k(z) \end{bmatrix} = \prod_{i=0}^{k-1} \begin{bmatrix} 1 & -\Delta_{k-i}z \\ -\Lambda_{k-i} & z \end{bmatrix} \tag{4.85}$$

so

$$\begin{bmatrix} Z_k^t(z) \\ \hat{V}_k^t(z) \end{bmatrix} = \begin{bmatrix} E_k(z) & zF_k(z) \\ G_k(z) & zH_k(z) \end{bmatrix} \begin{bmatrix} I_p \\ I_p \end{bmatrix} \tag{4.86}$$

At this point, we can use the divide-and-conquer technique to compute $A_n(z)$,$B_n(z)$, $C_n(z)$, $D_n(z)$, $E_n(z)$, $F_n(z)$, $G_n(z)$ and $H_n(z)$.

The outline of the superfast algorithm is shown below.

1. If $n = 1$, then

$$\begin{aligned} A_n(z) &= 1 \\ B_n(z) &= -S_{n-1,0}^{-1} R_{n-1,0} \\ C_n(z) &= Q_{n-1,0}^{-1} P_{n-1,0} \\ D_n(z) &= 1 \\ E_n(z) &= 1 \\ F_n(z) &= -R_{n-1,0} Q_{n-1,0}^{-1} \\ G_n(z) &= -P_{n-1,0} S_{n-1,0}^{-1} \\ H_n(z) &= 1 \end{aligned}$$

2. If $n > 1$, then call the algorithm to compute $A_{\frac{n}{2}}(z)$,$B_{\frac{n}{2}}(z)$, $C_{\frac{n}{2}}(z)$, $D_{\frac{n}{2}}(z)$, $E_{\frac{n}{2}}(z)$, $F_{\frac{n}{2}}(z)$, $G_{\frac{n}{2}}(z)$ and $H_{\frac{n}{2}}(z)$.

3. Compute $P_{\frac{n}{2}-1}(z)$,$Q_{\frac{n}{2}-1}(z)$, $R_{\frac{n}{2}-1}(z)$,$S_{\frac{n}{2}-1}(z)$. From (4.80)-(4.82) and (4.84), we have

$$\begin{bmatrix} P_{\frac{n}{2}-1}(z) & Q_{\frac{n}{2}-1}(z) \end{bmatrix} = \begin{bmatrix} P_{n-1}(z) & Q_{n-1}(z) \end{bmatrix} \begin{bmatrix} z^{-\frac{n}{2}} A_{\frac{n}{2}}(z) & z^{1-\frac{n}{2}} B_{\frac{n}{2}}(z) \\ z^{-\frac{n}{2}} C_{\frac{n}{2}}(z) & z^{1-\frac{n}{2}} D_{\frac{n}{2}}(z) \end{bmatrix}$$

$$\begin{bmatrix} R_{\frac{n}{2}-1}(z) & S_{\frac{n}{2}-1}(z) \end{bmatrix} = \begin{bmatrix} R_{n-1}(z) & S_{n-1}(z) \end{bmatrix} \begin{bmatrix} \frac{1}{z}D_{\frac{n}{2}}(\frac{1}{z}) & C_{\frac{n}{2}}(\frac{1}{z}) \\ \frac{1}{z}B_{\frac{n}{2}}(\frac{1}{z}) & A_{\frac{n}{2}}(\frac{1}{z}) \end{bmatrix}$$

4. Call the algorithm to compute $A^n_{\frac{n}{2}}(z)$,$B^n_{\frac{n}{2}}(z)$, $C^n_{\frac{n}{2}}(z)$, $D^n_{\frac{n}{2}}(z)$, $E^n_{\frac{n}{2}}(z)$, $F^n_{\frac{n}{2}}(z)$, $G^n_{\frac{n}{2}}(z)$ and $H^n_{\frac{n}{2}}(z)$.

5. Compute

$$\begin{bmatrix} A_n(z) & B_n(z) \\ zC_n(z) & zD_n(z) \end{bmatrix} = \begin{bmatrix} A_{\frac{n}{2}}(z) & B_{\frac{n}{2}}(z) \\ zC_{\frac{n}{2}}(z) & zD_{\frac{n}{2}}(z) \end{bmatrix} \begin{bmatrix} A^n_{\frac{n}{2}}(z) & B^n_{\frac{n}{2}}(z) \\ zC^n_{\frac{n}{2}}(z) & zD^n_{\frac{n}{2}}(z) \end{bmatrix}$$

and

$$\begin{bmatrix} E_n(z) & zF_n(z) \\ G_n(z) & zH_n(z) \end{bmatrix} = \begin{bmatrix} E^n_{\frac{n}{2}}(z) & zF^n_{\frac{n}{2}}(z) \\ G^n_{\frac{n}{2}}(z) & zH^n_{\frac{n}{2}}(z) \end{bmatrix} \begin{bmatrix} E_{\frac{n}{2}}(z) & zF_{\frac{n}{2}}(z) \\ G_{\frac{n}{2}}(z) & zH_{\frac{n}{2}}(z) \end{bmatrix}$$

6. Compute

$$\begin{aligned} X_n(z) &= A_n(z) + zC_n(z) \\ \hat{W}_n(z) &= B_n(z) + zD_n(z) \\ Z^t_n(z) &= E_n(z) + zF_n(z) \\ \hat{V}^t_n(z) &= G_n(z) + zH_n(z) \end{aligned}$$

Using (4.33), we can get the inversion of block Toeplitz matrix. In the algorithm, we use the FFT algorithm to compute the multiplication of two polynomials.

Studying the algorithm, we find that each stage of the algorithm requires 32 fast Fourier transforms (we take the inverse FFT same as the FFT). Since each block is $p \times p$, so we

need a total of $32p^2$ FFT. The number of arithmetic operations is

$$\begin{aligned} M(n) &= 2M(\frac{n}{2}) + 32p^2\frac{n}{2}log_2 n + O(p^3 n) \\ &= 8p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n) \\ A(n) &= 2A(\frac{n}{2}) + 32p^2 n log_2 n + O(p^3 n) \\ &= 16p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n). \end{aligned}$$

Whre $M(n)$ is the number of MULT and $A(n)$ is the number of ADD required by the algorithm.

There are two case which will simplify the computation.

1. The matrix $T_{p,n}$ is symmetric, i.e. $T_{p,n} = T'_{p,n}$. In this case, from (4.6) and (4.12), we find that

$$\begin{aligned} \alpha'_n &= T'_0 - \hat{\underline{T}}'_{-n}(T^{-1}_{p,n-1})'(\hat{\underline{T}}^t_n)' = \alpha_n \\ \beta'_n &= \beta_n \end{aligned}$$

Taking transpose of (4.41) and (4.43), we have

$$T^t_{p,n}(\underline{V}^t_n)' = \begin{bmatrix} \alpha_n \\ \\ 0 \end{bmatrix}$$

and

$$T_{p,n}(\underline{Z}^t_n)' = \begin{bmatrix} \beta_n \\ \\ 0 \end{bmatrix}$$

Comparing them with (4.40) and (4.42), we get

$$(\underline{V}^t_n)' = \underline{W}_n$$

and

$$(\underline{Z}_n^t)' = \underline{X}_n$$

2. The matrix $T_{p,n}$ is symmetric and each submatrix is Toeplitz, i.e. $T_{p,n}$ is Toeplitz–block–Toeplitz matrix.

This case has been studied in [18]. The result is

$$T_{p,n}\underline{X}_n = \begin{bmatrix} \alpha_n \\ 0 \end{bmatrix}$$

We can prove in this case, $A_k(x) = \hat{D}_k(x)$ and $B_k(x) = \hat{C}_k(x)$. The total number of arithmetic operations is

$$\begin{aligned} M(n) &= 2.5p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n) \\ A(n) &= 5p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n) \end{aligned}$$

Note in [18], $J_{p,n}$ is defined as

$$J_{p,n} = \begin{bmatrix} 0 & 0 & \cdots & 0 & J_p \\ 0 & 0 & \cdots & J_p & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ J_p & 0 & \cdots & 0 & 0 \end{bmatrix}$$

If $T_{p,n}$ is not symmetric but Toeplitz–block–Toeplitz, we have

$$\begin{aligned} T_{p,n}\underline{X}_n &= \begin{bmatrix} \beta_n \\ 0 \end{bmatrix} \\ T_{p,n}^t\underline{W}_n &= \begin{bmatrix} \alpha_n \\ 0 \end{bmatrix} \end{aligned}$$

The total operations are

$$M(n) = 4.5p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n)$$

$$A(n) = 9p^2 n log_2^2 n + O(p^2 n log_2 n) + O(p^3 n)$$

4.4 Conclusion

In this chapter, we studied the superfast and fast algorithms for computing the inverse of block Toeplitz matrices. The results are valid for any kind of block Toeplitz matrix. Some simplifications are also studied. The results are very efficient when $p \ll n$. It should be mentioned that Toeplitz–plus–Hankel matrix can be transformed into Toeplitz–block–Toeplitz matrix [19]. In this case, $p = 2$. So the superfast algorithm will be very efficient in this case. For the scalar Toeplitz matrix, de Hoog has studied the superfast algorithm [10]. In our case, $p = 1$ corresponds to the scalar case.

Chapter 5

The Finite Precision Analysis of Classical and Split Algorithms

The Levinson algorithm is widely used in the linear prediction because of its simple recursive and efficient structure. There are other algorithms that have also been used. These algorithms are Schur algorithm and lattice algorithm. Recently, the redundancy of these algorithms have been shown in [9]. In [9], a set of new algorithm defined as split Levinson algorithm, split Schur algorithm and split lattice algorithm have been proposed. The difference between classical and their split correspondent is that the split algorithms save computation time by almost 50%.

In spite of the wide use of those algorithms, a very important aspect has not been studied. This aspect is the effect of finite precision of computer. This problem is inevitable in the implementation oe the algorithms because of the finite word-length of computer.

The finite precision of algorithms is very important in the practical situations. For

example, a designed filter that is stable in theory, may becomw unstable after computations by the computer.

So far, some results about Levinson and lattice algorithm were reported. In [20] and [14], the authors showed that implementing Levinson algorithm using finite precision arithmetic results in instability. G. Cybenko analyzed the finite precision effect on the stability of Levinson algorithm. But he did not give a formula to compute the errors of reflection coefficients [6]. The recently report on finite precision effects on Levinson and lattice algorithm was done in [12]. They give two simple formula for computing errors due to quantization. However , they did not analyzed other algorithms mentioned.

In this chapter, we study the finite precision effects on Levinson, Schur, lattice and their split cases. Some simple but useful formulas will be given for estimation the errors in the refletion coefficients.

5.1 The Classical Algorithms

5.1.1 The Levinson Algorithm

In the previous chapter, we have studied Levinson algorithm. For convenience of the reader, we will briefly reintroduce it. For a sequence of real signal samples $u(0), u(1), \cdots, u(n-1)$ of finite length n, the autocorrelation are defined as

$$c_i = \sum_{t=0}^{n-i} u(t)u(t-i)$$

The k-th reflection coefficient can be computed by using the Levinson algorithm. The procedure of computation is as follows.

For $k = 1, 2, \cdots, n$

$$g_k = \sum_{i=0}^{k-1} c_{k-i} a_{k-1,i} \tag{5.1}$$

$$\rho_k = -\frac{g_k}{\alpha_{k-1}} \tag{5.2}$$

$$a_{k,i} = a_{k-1,i} + \rho_k a_{k-1,k-i} \tag{5.3}$$

$$\alpha_k = \alpha_{k-1} + \rho_k g_k \tag{5.4}$$

The initial conditions are $a_{0,0} = 1$, and $\alpha_0 = c_0$. In these formula, $a_{k,i}$ is a predictor coefficient, and α_k is a prediction error.

5.1.2 Schur Algorithm

Schur algorithm can be derived from the Levinson algorithm. This has been studied in the last chapter. In the following, we will reintroduce the scalar Schur algorithm which will be analysed in the later portions of the chapter.

In the last chapter, we obtained the general Schur algorithm, which is valid for a block general Toeplitz matrix. Since in this chapter, we are only dealing with the symmetric scalar Toeplitz matrix case, the general results can be simplified.

Defining the vectors p_{n-k}, q_{n-k} by

$$\begin{bmatrix} p_{n-k} & q_{n-k} \end{bmatrix} = \begin{bmatrix} o_k & I_{n-k} \end{bmatrix} T_n \begin{bmatrix} \underline{a}_k & 0 \\ 0 & \underline{\hat{a}}_k \\ \vdots & \vdots \end{bmatrix} \tag{5.5}$$

and computing $\underline{p}_{n-k}$ and $\underline{q}_{n-k}$, we have

$$p_{n-k-1,j} = \sum_{i=0}^{k} c_{k+1+j-i} a_{k,i} \tag{5.6}$$

$$q_{n-k-1,j} = \sum_{i=0}^{k} c_{i+j} a_{k,i} \tag{5.7}$$

Introducing the polynomials

$$p_{n-k}(z) = \sum_{i=0}^{n-k} p_{n-k,i} z^i$$

$$q_{n-k}(z) = \sum_{i=0}^{n-k} q_{n-k,i} z^i$$

and taking the same procedures as those in last chapter, we obtain the recurrence relation

$$z p_{n-k-1}(z) = \rho_k q_{n-k}(z) + p_{n-k}(z) \tag{5.8}$$

$$q_{n-k-1}(z) = q_{n-k}(z) + \rho_k p_{n-k}(z) \tag{5.9}$$

$$\rho_k = -\frac{p_{n-k,0}}{q_{n-k,0}} \tag{5.10}$$

In this chapter, we will introduce one set of variables instead of $\underline{p}_{n-k}$ and $\underline{q}_{n-k}$ by defining

$$h_{k,j} = \sum_{i=0}^{k} c_{|j-i|} a_{k,i} \tag{5.11}$$

where $k + 1 - n \leq j \leq n$.

Comparing with (5.6) and (5.7) , one can note that

$$h_{k,-j} = q_{n-k-1,j}$$

$$h_{k,j+k+1} = p_{n-k-1,j}$$

The Schur algorithm with $h_{k,j}$ is

$$\rho_k = -\frac{h_{k-1,k}}{h_{k-1,0}} \tag{5.12}$$

$$h_{k,-j} = h_{k-1,-j} + \rho_k h_{k-1,k+j} \tag{5.13}$$

$$h_{k,k+j+1} = \rho_k h_{k-1,-j-1} + h_{k-1,k+j+1} \tag{5.14}$$

5.1.3 Lattice Algorithm

Lattice algorithm [21] is different from the first two algorithms. In Levinson and Schur algorithms, we first calculate the autocorrelation coefficient of the data, then solve the Toeplitz matrix. In lattice algorithm, we directly compute the reflection coefficients and forward-backward prediction errors by using the signal samples.

The lattice algorithm can be implemented by a filter called lattice filter. This filter is widely used in the speech prediction. It is also the basic frame for ARMA model [11].

The two important terms in the lattice algorithm are the forward prediction error $\underline{f}_k$ and the bachward prediction error $\underline{b}_k$. For the given signal u_0, u_1, $\cdots$, u_{n-1}, we can form the forward prediction error of order k or bachward prediction error of order k. and the forward prediction error is defined as

$$f_{k,j} = \sum_{i=0}^{k} a_{k,i} u_{j-i}$$

The backward prediction error is defined as

$$b_{k,j} = \sum_{i=0}^{k} a_{k,k-i} u_{j-i}$$

Multiplying by u_{j-i} on both side of Levinson algorithm in (5.3) and the reciprocal Levinson algorithm and summing the result from $i = 0$ to $i = k$, one gets the lattice algorithm.

The lattice algorithm is summarized as follows.

For $k = 1, 2, \cdots n$

$$g_k = -\sum_{i=1}^{n+k-2} f_{k-1,i} b_{k-1,i-1} \tag{5.15}$$

$$\rho_k = \frac{g_k}{\alpha_{k-1}} \tag{5.16}$$

$$f_{k,i} = f_{k-1,i} + \rho_k b_{k-1,i-1} \tag{5.17}$$

$$b_{k,i} = \rho_k f_{k-1,i} + b_{k-1,i-1} \tag{5.18}$$

The initial conditions are $f_{0,i} = b_{0,i} = u_i$ and $\alpha_0 = \sum_{i=0}^{n-1} u_i^2$.

We also can express the lattice algorithm in the polynomial form. Let us introduce two polynomials which are defined as

$$f_{n+k-1}(z) = \sum_{i=0}^{n+k-1} f_{k,i} z^i$$

$$b_{n+k-1}(z) = \sum_{i=0}^{n+k-1} b_{k,i} z^i$$

Substituting the polynomials into lattice algorithm (5.17) and (5.18), we have

$$f_{n+k-1}(z) = f_{n+k-2}(z) + \rho_k z b_{n+k-2}(z) \tag{5.19}$$

$$b_{n+k-1}(z) = \rho_k f_{n+k-2}(z) + z b_{n+k-2}(z) \tag{5.20}$$

The initial polynomials are

$$b_{n-1}(z) = f_{n-1}(z) = \sum_{i=0}^{n-1} u_i z^{n-1}$$

5.2 Split Algorithms

5.2.1 Split Levinson Algorithm

In the previous studies, we introduced the Levinson, Schur and lattice algorithms. They are called the classical algorithms. The computational complexity of these classical algorithms is $O(n^2)$.

Recently, more efficient algorithms were proposed [9]. These are named split algorithms. The main idea of split algorithms is to reduce the redundancy existing in the classical algorithms. This can be done by introducing a set of symmetric or antisymmetric sequence. As we know, every function is a combination of its symmetric and antisymmetric parts. Attributed to the beautiful structure of Toeplitz matrix, the Toeplitz problem can be solved only by the symmetric or antisymmetric parts. Since the only half the coefficients need to specify the problem, the complexity of split algorithms will be reduced by one-half time compared to their corresponding classical algorithms.

we have met the split Levinson algorithm in Chapter 3. In the following, from the polynomial, we will study some details of the algorithm.

Let us define the symmetric polynomial by

$$s_k(z) = a_{k-1} + z^{-1}\hat{a}_{k-1}(z) \tag{5.21}$$

The polynomial $s_k(z)$ are symmetric since

$$\begin{aligned} \hat{s}_k &= z^{-k}[a_{k-1}(z^{-1}) + z\hat{a}_{k-1}(z^{-1})] \\ &= z^{-1}\hat{a}_{k-1}(z) + a_{k-1}(z) \end{aligned}$$

Note that $s_k(z)$ is a special $a_k(z)$ which corresponds to the reflection coefficient ρ_k equals one. Under this situation, the Toeplitz matrix is singular.

The antisymmetric polynomial $t_k(z)$ can be defined as

$$t_k(z) = a_{k-1}(z) - z^{-1}\hat{a}_{k-1}(z)$$

$t_k(z)$ is antisymmetric since

$$\begin{aligned} \hat{t}(z) &= z^{-k}[a_{k-1}(z^{-1}) - z\hat{a}_{k-1}(z^{-1})] \\ &= z^{-1}\hat{a}_{k-1}(z) - a_{k-1}(z) \\ &= -t_k(z) \end{aligned}$$

We will focus our attention to the symmetric case.

The Levinson algorithm can be written in polynomial form as

$$a_k(z) = a_{k+1}(z) + z^{-1}\rho_k\hat{a}_{k-1}(z) \tag{5.22}$$

The reciprocal polynomial of $a_k(z)$ derived from (5.22) is

$$\hat{a}_k(z) = z^{-1}\hat{a}_{k-1}(z) + \rho_k a_{k-1}(z)$$

Substituting for $a_k(z)$ from (5.22) and $\hat{a}_k(z)$ from above in (5.21), one obtains

$$(1+\rho_k)s_k(z) = a_k(z) + \hat{a}_k(z) \tag{5.23}$$

Incrementing the index in (5.21), one gets,

$$s_{k+1}(z) = a_k(z) + z^{-1}\hat{a}_k(z)$$

Multiplying both side of (5.23) by z^{-1} and taking the difference with $s_{k+1}(z)$, we get,

$$(1 - z^{-1})a_k(z) = s_{k+1}(z) - (1 + \rho_k)z^{-1}s_k(z) \tag{5.24}$$

At this point we heve established the relation between the $a_k(z)$ and $s_k(z)$. By using those relations, we can derive a recurrence on $s_k(z)$ instead of the Levinson algorithm.

The derivation of the recurrence of $s_k(z)$ as same as the derivation of stepdown procedure in Chapter 2. We directly write the relation as

$$s_{k+1}(z) = (1 + z^{-1})s_k(z) + \beta_k z^{-1}s_{k-1}(z) \tag{5.25}$$

where $\beta_k = (1 - \rho_k)(1 + \rho_{k-1})$ The initial polynomial of $s_k(z)$ are

$$s_0(z) = 2, \qquad s_1(z) = 1 + z^{-1}$$

The split Levinson algorithm also can be written as

$$\tau_k = \sum_{i=0}^{k} c_i s_{k,i} \tag{5.26}$$

$$\beta_k = \frac{\tau_k}{\tau_{k-1}} \tag{5.27}$$

$$s_{k+1,i} = s_{k,i} + s_{k,i-1} - \beta_k s_{k-1,i-1} \tag{5.28}$$

$$\rho_k = 1 - \frac{\beta_k}{1 + \rho_{k-1}} \tag{5.29}$$

with the conditions $s_{k,0} = 1$, $(k \geq 1)$, $s_{0,0} = 2$.

5.2.2 Split Schur Algorithm

The split Schur algorithm can be dericed from the Schur algorithm by defining

$$v_{k,j} = h_{k-1,-j} + h_{k-1,k+j} \tag{5.30}$$

or

$$y_{n-k}(z) = q_{n-k}(z) + p_{n-k}(z) \tag{5.31}$$

where

$$y_{n-k}(z) = v_{k,0} + v_{k,1}z + \cdots + v_{k,n-k}x^{n-k}$$

From (5.31), we have

$$(1+\rho_k)y_{n-k}(z) = q_{n-k}(z) + \rho_k p_{n-k}(z) + \rho_k q_{n-k}(z) + p_{n-k}(z)$$

Using Schur algorithm in (5.8) and (5.9), we have

$$(1+\rho_k)y_{n-k}(z) = q_{n-k-1}(z) + zp_{n-k-1}(z) \bmod z^{n-k} \tag{5.32}$$

Reducing the index k in (5.32) by one, one have

$$(1+\rho_{k-1})y_{n-k-1}(z) = q_{n-k}(z) + zp_{n-k}(z) \bmod x^{n-k+1} \tag{5.33}$$

Substracting (5.31) by (5.33), and substracting $zy_{n-k}(z)$ by (5.33), we have

$$y_{n-k}(z) - (1+\rho_{k-1})y_{n-k+1}(z) = (1-z)p_{n-k}(z) \bmod z^{n-k+1} \tag{5.34}$$

$$zy_{n-k}(z) - (1+\rho_{k-1})y_{n-k+1}(z) = (z-1)q_{n-k}(z) \bmod z^{n-k+1} \tag{5.35}$$

At this point, we are going to derive the split Schur algorithm from Schur algorithm. We have known Schur algorithm by (5.8) and (5.9). Multiplying $(1-z)$ to (5.8), we have

$$(1-z)q_{n-k-1}(z) = (1-z)q_{n-k}(z) + (1-z)\rho_k p_{n-k}(z) \bmod z^{n-k+1}$$

Using (5.34) and (5.35) in the results, we have

$$\begin{aligned} -zy_{n-k-1}(z) + (1+\rho_k)y_{n-k}(z) &= -zy_{n-k}(z) + (1+\rho_{k-1})y_{n-k+1}(z) \\ &\quad +\rho_k(y_{n-k}(z) - (1+\rho_{k-1})y_{n-k+1}(z) \bmod z^{n-k+1} \end{aligned}$$

rearranging the terms on both side, we get

$$-zy_{n-k-1}(z) = -(1+z)y_{n-k}(z) + (1-\rho_k)(1+\rho_{k-1})y_{n-k+1}(z) \bmod z^{n-k+1}$$

Finally we can obtain the Schur algorithm after removing the factor z on both side of the last equation. The Schur algorithm has the form

$$y_{n-k-1}(z) = (1+z^{-1})y_{n-k}(z) - \beta_k z^{-1} y_{n-k+1}(z) \bmod z^{n-k} \tag{5.36}$$

where

$$\begin{aligned} \beta_k &= (1-\rho_k)(1+\rho_{k-1}) \\ &= \frac{v_{k,0}}{v_{k-1,0}} \end{aligned}$$

The initials of the polynomial are

$$y_n(z) = c_0 + \sum_{i=1}^{n} 2c_i z^{-i} \tag{5.37}$$

$$y_{n-1}(z) = \sum_{i=0}^{n-1} (c_i + c_{i+1}) z^{-1} \tag{5.38}$$

We also write the Schur algorithm in the form as

For $k = 1, 2, \cdots n$

$$\beta_k = \frac{v_{k,0}}{v_{k-1,0}} \tag{5.39}$$

$$\rho_k = 1 - \frac{\beta_k}{1+\rho_{k-1}} \tag{5.40}$$

$$v_{k+1,j} = v_{k,j} + v_{k,j+1} - \beta_k v_{k-1,j+1} \tag{5.41}$$

5.2.3 Split Lattice Algorithm

Taking the same procedure as in previous sections, we can derive the split lattice algorithm from the lattice algorithm.

Let

$$w_{k,j} = f_{k-1,j} + b_{k-1,j-1} \tag{5.42}$$

or

$$w_{n+k-1}(z) = f_{n+k-2}(z) + zb_{n+k-2}(z) \tag{5.43}$$

Using the lattice algorithm in (5.19) and (5.20), multiplying $(1 + \rho_k)w_{n+k-1}(z)$, we have

$$(1 + \rho_k)w_{n+k-1}(z) = f_{n+k-1}(z) + b_{n+k-1}(z) \tag{5.44}$$

Incrementing the index k in (5.43) by one, and subtracting it with (5.44), we have

$$w_{n+k}(z) - (1 + \rho_k)w_{n+k-1}(z) = (z - 1)b_{n+k-1}(z) \tag{5.45}$$

Subtract (5.43) with incrementing k by one from (5.44), we have

$$w_{n+k}(z) - (1 + \rho_k z w_{n+k-1}(z) = (1 - z)f_{n+k-1}(z) \tag{5.46}$$

These relations built up the bridge between $w_{n+k}(z)$ and $b_{n+k}(z)$ and $f_{n+k}(z)$. They will be used to establish the split lattice algorithm.

The split lattice algorithm can be obtained by multiplying (5.19) by $(1 - z)$ and using (5.45) and (5.46). The result is shown below

$$\begin{aligned} w_{n+k}(z) - (1 + \rho_k)zw_{n+k-1}(z) \;=\; & w_{n+k-1}(z) - (1 + \rho_{k-1})zw_{n+k-2}(z) \\ & +\rho_k z((1 + \rho_{k-1})w_{n+k-2}(z) - w_{n+k-1}(z)) \end{aligned}$$

Simplifying, it we have the split lattice algorithm as

$$w_{n+k}(z) = w_{n+k-1}(z)(1+z) - \beta_k z w_{n+k-1}(z) \tag{5.47}$$

where $\beta_k = (1-\rho_k)(1+\rho_{k-1})$. The initial polynomial are

$$\begin{aligned} w_{n-1}(z) &= \sum_{i=0}^{n-1} 2u_i z^i \\ w_n(z) &= \sum_{i=0}^{n} (u_i + u_{i-1}) z^i \end{aligned}$$

The split lattice algorithm can be also written in the form as

For $k = 1, 2, \cdots n$,

$$2\tau_k = \sum_{i=0}^{n+k-1} w_{k,i}^2 \tag{5.48}$$

$$\beta_k = \frac{\tau_k}{\tau_{k-1}} \tag{5.49}$$

$$\rho_k = 1 - \frac{\beta_k}{1+\rho_{k-1}} \tag{5.50}$$

$$w_{k+1,i} = w_{k,i} + w_{k,i-1} - \beta_k w_{k-1,i-1} \tag{5.51}$$

The initial conditions are

$$\begin{aligned} w_{0,i} &= 2u_i \\ w_{1,i} &= u_i + u_{i-1} \end{aligned}$$

5.3 Finite Precision Analysis of Classical Algorithms

5.3.1 Finite Analysis of Levinson Algorithm

The finite wordlength of computer gives rise to the need to analyse the precision of output by computer. Due to the finite wordlength of computer, some errors are caused in the

results obtained by performing numerical computations. If the errors grow too large, the algorithm can not be used.

For analysing the finite precision effects of computer on the results, we should explore the structure of algorithms. Different algorithms will react in a different manner. Some of them give up a precise enough result; some of them not. For the analysis, we first establish the model used in the chapter. Let x represent a precise variable, then after quantization it will become

$$\tilde{x} = x + e_x$$

where the $\tilde{x}$ represents the approximation of the variable x, and e_x is the quantization error.

In this report, we only study the fixed point arithmetic. In this case, since the output of two b-bit number multiplying or dividing is a 2b-bits number, the quantization errors will occur. For the addition and subtraction, there is no error created under the condition of no overflow.

After quantization, the Levinson algorithm becomes

$$\tilde{g}_k = \sum_{i=0}^{k-1} \tilde{c}_{k-i}\tilde{a}_{k-1,i} + e_{g,k} \tag{5.52}$$

$$\tilde{\rho}_k = -\frac{\tilde{g}_k}{\tilde{\alpha}_{k-1}} - e_{\rho,k} \tag{5.53}$$

$$\tilde{a}_{k,i} = \tilde{a}_{k-1,i} + \tilde{\rho}_k\tilde{a}_{k-1,k-i} + e_{a,k,i} \tag{5.54}$$

$$\tilde{\alpha}_k = \tilde{\alpha}_{k-1} + \tilde{\rho}_k\tilde{g}_k + e_{\alpha,k} \tag{5.55}$$

with the conditions $a_{0,0} = 1$ and $\alpha_0 = c_0$.

The errors of the computation are obtained by subtraction (5.1)-(5.4) from (5.52)-(5.55).

For example, taking (5.1) away from (5.52), we get

$$\begin{aligned}
\Delta g_k &= \tilde{g}_k - g_k \\
&= \sum_{i=0}^{k-1} \tilde{c}_{k-i}\tilde{a}_{k-1,i} + e_{g,k} - \sum_{i=0}^{k-1} c_{k-i}a_{k-1,i} \\
&= \sum_{i=0}^{k-1} (\tilde{c_{k-i}}\tilde{a}_{k-1,i} - c_{k-i}\tilde{a}_{k-1,i} + c_{k-i}\tilde{a}_{k-1,i} - c_{k-i}a_{k-1,i}) + e_{g,k} \\
&= \sum_{i=0}^{k-1} (\tilde{a}_{k-1,i}\Delta c_{k-i} + c_{k-i}\Delta \tilde{a}_{k-1,i}) + e_{g,k} \\
&= \sum_{i=0}^{k-1} (a_{k-1,i}\Delta c_{k-i} + c_{k-i}\Delta a_{k-1,i}) + e_{g,k} \qquad (5.56)
\end{aligned}$$

Where we have defined $\Delta()$ being an operator as $\Delta x = \tilde{x} - x$.

In the last step, we use $a_{k-1,i}\Delta c_{k-i}$ to replace $\tilde{a}_{k-1,i}\Delta c_{k-i}$. Because $\tilde{a}_{k-1,i} = a_{k-1,i} + \Delta a_{k-1,i}$, so $\tilde{a}_{k-1,i}\Delta c_{k-i} = a_{k-1,i}\Delta c_{k-i} + \Delta a_{k-1,i}\Delta c_{k-i}$. The last term are the product of two operator Δ, that is , $\Delta()\Delta()$. We call such terms second order errors. In this report, we assume that the second or higher order errors are much smaller than the first order, and therefore, they are ignored.

Taking similar procedure on the other quantities in Levinson algorithm, we have

$$\begin{aligned}
\Delta \rho_k &= \tilde{\rho}_k - \rho_k \\
&= -\frac{\rho_k\Delta \alpha_{k-1} + \Delta g_k}{\alpha_{k-1} + \Delta \alpha_{k-1}} - e_{\rho,k}
\end{aligned}$$

Since $\alpha_{k-1} \gg \Delta \alpha_{k-1}$, we can simplify the result as

$$\Delta \rho_k = -\frac{\rho_k\Delta \alpha_{k-1} + \Delta g_k}{\alpha_{k-1}} - e_{\rho,k} \qquad (5.57)$$

$$\begin{aligned}
\Delta a_{k,i} &= \tilde{a}_{k,i} - a_{k,i} \\
&= \Delta a_{k-1,i} + \rho_k\Delta a_{k-1,k-i} + a_{k-1,k-i}\Delta \rho_k + e_{a,k,i} \qquad (5.58)
\end{aligned}$$

Repeating the relation $\Delta a_{k,i}$ to $\Delta a_{k-1,i}$, $\Delta a_{k-2,i}$, $\cdots$, we have

$$\Delta a_{k,i} = \sum_{j=1}^{k} (\rho_j \Delta a_{j-1,j-i} + a_{j-1,j-i} \Delta \rho_j + e_{a,j,i}) \tag{5.59}$$

For α_k, we have

$$\begin{aligned} \Delta \alpha_k &= \tilde{\alpha_k} - \alpha_k \\ &= \Delta \alpha_{k-1} + \rho_k \Delta g_k + g_k \Delta \rho_k + e_{\alpha,k} \end{aligned} \tag{5.60}$$

From (5.57), we have known

$$\Delta g_k = -\alpha_{k-1} \Delta \rho_k - \rho_k \Delta \alpha_{k-1} - \alpha_{k-1} e_{\rho,k}$$

Substituting Δg_k into (5.60), we have

$$\Delta \alpha_k = (1 - \rho^2) \Delta \alpha_{k-1} - \rho_k \alpha_{k-1} \Delta \rho_k - \rho_k \alpha_{k-1} e_{\rho,k} + g_k \Delta \rho_k + e_{\alpha,k}$$

Noting that $-\rho_k \alpha_{k-1} = g_k$, we have

$$\Delta \alpha_k = (1 - \rho_k) \Delta \alpha_{k-1} + 2 g_k \Delta \rho_k + g_k e_{\rho,k} + e_{\alpha,k} \tag{5.61}$$

Continuing to expand $\Delta \alpha_{k-2}$, $\Delta \alpha_{k-3}$, $\cdots$, $\Delta \alpha_0$, we have

$$\begin{aligned} \Delta \alpha_k &= \prod_{i=1}^{k} (1 - \rho_i^2) \Delta \alpha_0 + \sum_{i=0}^{k-1} \prod_{j=i}^{k-1} (1 - \rho_{j+1}^2) \left[2 g_i \Delta \rho_i + e_{\alpha,i} + g_i e_{\rho,i} \right] \\ &\quad + 2 g_k \Delta \rho_k + e_{\alpha,k} + g_k e_{\rho,k} \end{aligned}$$

Since $\alpha_j = \prod_{i=1}^{k} (1 - \rho_i^2) \alpha_0$, we can simplify $\Delta \alpha_k$ into

$$\begin{aligned} \Delta \alpha_k &= \frac{\alpha_k}{\alpha_0} \Delta \alpha_0 + \sum_{i=0}^{k-1} \frac{\alpha_k}{\alpha_i} \left[2 g_i \Delta \rho_i + e_{\alpha,i} + g_i e_{\rho,i} \right] + 2 g_k \Delta \rho_k + e_{\alpha,k} + g_k e_{\rho,k} \\ &= \frac{\alpha_k}{\alpha_0} \Delta \alpha_0 + \alpha_k \sum_{i=0}^{k} \left(- 2 \rho_i \Delta \rho_i - \rho_i e_{\rho,i} + \frac{e_{\alpha,i}}{\alpha_i} \right) \end{aligned} \tag{5.62}$$

Substituting $\Delta\alpha_k$ into $\Delta\rho_k$ as in (5.57), we finally get the error $\Delta\rho_k$ as

$$\Delta\rho_k = -\rho_k\frac{\Delta\alpha_0}{\alpha_0} + \rho_k\sum_{i=0}^{k-1}\left(2_{\rho_i}\Delta\rho_i + \rho_i e_{\rho,i} - \frac{e_{\alpha,i}}{\alpha_i}\right) - \frac{\Delta g_k}{\alpha_{k-1}} - e_{\rho,k} \tag{5.63}$$

At this point we disucss the result of (5.63). The complex result can be simplified. The simplest result on the error can be obtained if we neglect the term $\Delta\alpha_{k-1}$, $e_{g,k}$, $e_{\rho,k}$, $e_{a,j,i}$ in all derivation and $\Delta a_{j-1,j-i}$ in (5.59), we find out that

$$\begin{aligned}\Delta\rho_k &= -\frac{\Delta g_k}{\alpha_{k-1}} \\ &= -\frac{1}{\alpha_{k-1}}\sum_{i=0}^{k-1}\left(a_{k-1,i}\Delta c_{k-i} + c_{k-i}\sum_{j=1}^{k-1}a_{j-1,j-i}\Delta\rho_j\right)\end{aligned} \tag{5.64}$$

This formula is same as the result given in [12].

For more accuracy we can continue to expand $\Delta a_{k,i}$ as

$$\begin{aligned}\Delta a_{k,i} &= \sum_{j=1}^{k}\rho_i\sum_{j_1=1}^{j-1}(\rho_{j_1}\Delta a_{j_1-1,j_1-j+i} + a_{j_1-1,j_1-j+i}\Delta\rho_{j_1} + e_{a,j_1,j-i}) \\ &\quad + \sum_{j=1}^{k}(a_{j-1,j-i}\Delta\rho_j + e_{a,j,i})\end{aligned}$$

Continuing expansion of $\Delta a_{k,i}$, we find that $\Delta a_{k,i}$ is a first order function of terms ρ_j, $\rho_j\rho_{j_1}$, $\cdots$ and $\Delta\rho_j$, that is

$$\Delta a_{k,i} = \sum_{j=1}^{k} f(\rho_j, \rho_j\rho_{j_1}, \cdots)\Delta\rho_j \tag{5.65}$$

where

$$f(\rho_j, \rho_j\rho_{j_1}, \cdots) = f_0 + f_1\rho_j + f_2\rho_j\rho_{j_1} + \cdots$$

If we ignore the terms ρ_j, $\rho_j\rho_{j_1}$, $\cdots$ and $\Delta\rho_j$, that is, perform the first order approximation, we have

$$\Delta\rho_k = -\rho_k\frac{\Delta\alpha_0}{\alpha_0} - \frac{1}{\alpha_{k-1}}\sum_{i=0}^{k-1}\left[a_{k-1,i}\Delta c_{k-i} + c_{k-i}\sum_{j=1}^{k-1}(a_{j-1,j-i}\Delta\rho_j + e_{a,j,i})\right] - e_{\rho,k} \tag{5.66}$$

If we do not count the $e_{\rho,k}$,$e_{a,j,i}$ and $\Delta\alpha_0$,we end up with the (5.64).

For more accuracy, we give the second order approximation of the $\Delta\rho_k$ as

$$\Delta\rho_k = \frac{1}{\alpha_{k-1}}\sum_{i=0}^{k-1} c_{k-i}\sum_{j=1}^{k-1}\left[a_{j-1,j-i}\Delta\rho_j + e_{a,j,i} + \rho_j\sum_{j_1=1}^{j-1}(a_{j_1,j_1-j+i}\Delta\rho_{j_1} + e_{a,j_1,j-i})\right]$$
$$-\rho_k\frac{\Delta\alpha_0}{\alpha_0} - \rho_k\sum_{i=0}^{k-1}\frac{e_{\alpha,i}}{\alpha_i} - \frac{1}{\alpha_{k-1}}\sum_{i=0}^{k-1} a_{k-1,i}\Delta c_{k-i} \tag{5.67}$$

The results are staggeringly complex for continuing this procedure. But we can see that the errors in reflection coefficients at kth step are dependent on the errors in previous steps. Fortunatly, we do not need to pursue the higher order approximation in practice since the first two terms dominate the errors.

5.3.2 Finite Analysis of Schur Algorithm

In this subsection, we are going to continue the study on the finite analysis for the Schur algorithm. Schur algorithm is given in section 5.1.2. Since the erors are introduced by the finite wordlength of computer, the algorithm has the following form after quantization

$$\tilde{\rho}_k = -\frac{\tilde{h}_{k-1,k}}{\tilde{h}_{k-1,0}} - e_{\rho,k} \tag{5.68}$$

$$\tilde{h}_{k,-j} = \tilde{h}_{k-1,-j} + \tilde{\rho}_k\tilde{h}_{k-1,k+j} + e_{h,k,-j} \tag{5.69}$$

$$\tilde{h}_{k,k+j+1} = \tilde{\rho}_k\tilde{h}_{k-1,-j-1} + \tilde{h}_{k-1,k+j+1} + e_{h,k,k+j+1} \tag{5.70}$$

Taking the same procedure as in the Levinson algorithm, we have

$$\Delta\rho_k = -\frac{\Delta h_{k-1,k} + \rho_k\Delta h_{k-1,0}}{h_{k-1,0}} - e_{\rho,k} \tag{5.71}$$

$$\Delta h_{k,-j} = \Delta h_{k-1,-j} + h_{k-1,k+j}\Delta\rho_k + \rho_k\Delta_{k-1,k+j} + e_{k-1,k+j} \tag{5.72}$$

$$\Delta h_{k,k+j+1} = \Delta h_{k-1,k+j+1} + h_{k-1,-j-1}\Delta\rho_k + \rho_k\Delta h_{k-1,-j-1} \tag{5.73}$$

We expand (5.72) and (5.73) continuously until

$$\Delta h_{k,-j} = \Delta h_{0,-j} + \sum_{i=1}^{k} (h_{i-1,i+j}\Delta \rho_i + \rho_i \Delta h_{i-1,i+j} + e_{h,i,-j}) \tag{5.74}$$

$$\begin{aligned}\Delta h_{k,k+j+1} &= \Delta_{0,k+j+1} \\ &\quad + \sum_{i=1}^{k} (h_{i-1,-h-k+i-1}\Delta \rho_i + \rho_i \Delta h_{i-1,-j-k+i-1} + e_{h,i,k+j+1})\end{aligned} \tag{5.75}$$

is reached.

Let us compute $\Delta h_{k-1,0}$. Setting $j = 0$ in (5.69) and in (5.13) we have

$$\begin{aligned}\tilde{h}_{k,0} &= \tilde{h}_{k-1,0} + \tilde{\rho}_k \tilde{h}_{k-1,k} + e_{h,k,0} \\ h_{k,0} &= h_{k-1,0} + \rho_k h_{k-1,k}\end{aligned}$$

The error in $h_{k,0}$ is derived as

$$\Delta h_{k,0} = \Delta h_{k-1,0} + \rho_k \Delta h_{k-1,k} + h_{k-1,k}\Delta \rho_k + e_{h,k,0}$$

From (5.71), we have $\Delta h_{k-1,k}$. Substituting it into $\Delta h_{k,0}$, we get

$$\Delta h_{k,0} = \left(1 - \rho^2\right) \Delta h_{k-1,0} - \rho_k h_{k-1,0}\Delta \rho_k - \rho_k h_{k-1,0} e_{\rho,k} + h_{k-1,k}\Delta \rho_k + e_{h,k,0}$$

Repeating the formula again on $\Delta h_{k-1,0}$, $\Delta h_{k-2,0}$, $\cdots$, we finally have

$$\Delta h_{k,0} = \frac{h_{k,0}}{h_{0,0}}\Delta h_{0,0} + \sum_{i=0}^{k} h_{k,0} \left(-2\rho_i \Delta \rho_i - \rho_i e_{\rho,i} + \frac{e_{h,i,0}}{h_{i,0}}\right) \tag{5.76}$$

So the error $\Delta \rho_k$ for Schur algorithm is

$$\Delta \rho_k = -\frac{\rho_k}{h_{0,0}}\Delta h_{0,0} + \rho_k \sum_{i=0}^{k-1} (2\rho_i \Delta \rho_i + \rho_i e_{\rho,i}) - \rho_k \sum_{i=0}^{k-1} \frac{e_{h,i,0}}{h_{i,0}} - \frac{\Delta h_{k-1,k}}{h_{k-1,0}} - e_{\rho,k} \tag{5.77}$$

Since the formula is very complicated, we are going to simplify it. Firstly, we note, from(5.74) and (5.75), that $\Delta h_{k,-j}$ and $\Delta h_{k,k+j+1}$ have the form

$$\begin{aligned}\Delta h_{k,-j} &= \sum_{j=1}^{k} f_1(\rho_j, \rho_j\rho_{j_1}, \cdots)\Delta\rho_j \\ \Delta h_{k,k+j+1} &= \sum_{j=1}^{k} f_2(\rho_j, \rho_j\rho_{j_1}, \cdots)\Delta\rho_j\end{aligned}$$

If we only consider the terms that are not dependent on $\rho_j, \rho_j\rho_{j_1}, \cdots$, we have

$$\Delta h_{k-1,k} = \Delta h_{0,k} + \sum_{i=1}^{k-1}(h_{i-1,-k+i}\Delta\rho_i + e_{h,i,k})$$

Then subsituting $\Delta h_{k-1,k}$ into (5.77) , we get the first order approximation of $\Delta\rho_k$ as

$$\Delta\rho_k = -\frac{1}{h_{k-1,0}}\left[\Delta h_{0,k} + \sum_{i=1}^{k-1}(h_{i-1,-k+i}\Delta\rho_i + e_{h,i,k})\right] - e_{\rho,k} \tag{5.78}$$

This formula is much simpler than the one in Levinson algorithm (5.64).

We should note that $\Delta h_{0,k} = \Delta c_k$ and $h_{k-1,0} = \alpha_{k-1}$. How about $\sum_{i=1}^{k-1} h_{i-1,-k+i}\Delta\rho_i$?

Firstly, from (5.11), we write

$$h_{i-1,-k+i} = \sum_{j=0}^{i-1} c_{j+k-i}a_{i-1,j} \ .$$

So

$$\begin{aligned}\sum_{i=1}^{k-1} h_{i-1,-k+i}\Delta\rho_i &= \sum_{i=1}^{k-1}\sum_{j=0}^{i-1} c_{j+k-i}a_{i-1,j}\Delta\rho_i \\ &= \sum_{i=1}^{k-1}\sum_{j=1}^{i} c_{k-j}a_{i-1,i-j}\Delta\rho_i \\ &= \sum_{i=1}^{k-1} c_{k-i}\sum_{j=1}^{k-1} a_{j-1,j-i}\Delta\rho_j\end{aligned}$$

This result has shown that the coefficient of $\Delta\rho_i$ in the first order approximation of $\Delta\rho_k$ of Levinson and Schur algorithm are equal.

For more accuracy we can include the terms of ρ_i as the second order approximation. From (5.74) and (5.75), we have

$$\Delta h_{k-1,k} = \Delta h_{0,k} + \sum_{i=1}^{k-1} \left[h_{i-1,-j+k+i-1} \Delta \rho_i + \rho_i \left(\Delta h_{0,-k+i} + \sum_{i_1=1}^{i-1} h_{i_1-1,i_1-i+k} \Delta \rho_{i_1} + e_{h,i,k-i} \right) + e_{h,i,k} \right]$$

Substituting $\Delta h_{k-1,k}$ into (5.77), we have

$$\begin{aligned} \Delta \rho_k &= -\frac{1}{h_{k-1,i}} \sum_{i=0}^{k-1} \rho_i \left[\Delta c_{k-i} + \sum_{i_1}^{i-1} h_{i_1-1,i_1-i+k} \Delta \rho_{i_1} + e_{h,i_1,k-i} \right] - \frac{\rho_k}{h_{0,0}} \Delta h_{0,0} \\ &\quad + \Delta \rho_{k,0} - \frac{\rho_k}{h_{0,0}} \Delta h_{0,0} - \rho_k \sum_{i=0}^{k-1} \frac{e_{h,i,0}}{h_{i,0}} \end{aligned} \tag{5.79}$$

where $\Delta \rho_{k,0}$ is the first order approximation of $\Delta \rho_k$ given in (5.78).

Now we will prove that

$$\sum_{i=1}^{k-1} \rho_i \sum_{i_1=1}^{i-1} h_{i_1-1,i_1-i+k} \Delta \rho_{i_1} = \sum_{i=0}^{k-1} \rho_j \sum_{j_1=1}^{j-1} (a_{j_1-1,j_1-j+i} \Delta \rho_{j_1})$$

Let $j_{1-j+i} = u$, then

$$\sum_{i=1}^{k-1} c_{k-i} \sum_{j=1}^{k-1} \rho_j \sum_{j_1=1}^{j-1} (a_{j_1-1,j_1-j+i} \Delta \rho_{j_1}) = \sum_{j=1}^{k-1} \rho_j \sum_{j_1=1}^{j-1} \sum_{u=j_1-j}^{k-1+j_1-j} a_{j_1-1,u} c_{k-u+j_1-j} \Delta \rho_{j_1}$$

Since $a_{j_1-1,u} = 0$ if $u > j_1 - 1$ or $u < 0$, we have

$$\sum_{u=j_1-j}^{k-1+j_1-j} a_{j_1-1,u} c_{k-u+j_1-j} = \sum_{u=0}^{j_1-1} a_{j_1-1,u} c_{k-u+j_1-j}$$

Using (5.11), we proved the equality.

From previous study, we found that the cofficients of $\Delta \rho_i$ of the errors for Schur and Levinson algorithms are same. This property can be proved by induction. This also means that if we presume c_i is correct, i.e. $\Delta c_i = 0$, then the variance of errors $\Delta \rho_k$ are same

for both Schur and Levinson algorithm. Since the coefficiets of Δc_i are different in the two algorithms, in general, we can not conclude that both the algorithms provide same accuracy.

5.3.3 Finite Analysis of Lattice Algorithm

For the lattice algorithm, we take the same procedures as in previous sections. The lattice algorithm after quantization becomes

$$\tilde{g}_k = \sum_{i=1}^{n+k-2} \tilde{f}_{k-1,i}\tilde{b}_{k-1,i-1} + e_{g,k} \tag{5.80}$$

$$\tilde{\rho}_k = -\frac{\tilde{g}_k}{\tilde{\alpha}_{k-1}} - e_{\rho,k} \tag{5.81}$$

$$\tilde{f}_{k,i} = \tilde{f}_{k-1,i} + \tilde{\rho}_k\tilde{b}_{k-1,i-1} + e_{f,k,i} \tag{5.82}$$

$$\tilde{b}_{k,i} = \tilde{\rho}_k\tilde{f}_{k-1,i} + \tilde{b}_{k-1,i-1} + e_{b,k,i} \tag{5.83}$$

$$\tilde{\alpha}_k = \tilde{\alpha}_{k-1} + \tilde{g}_k\tilde{\rho}_k + e_{\alpha,k} \tag{5.84}$$

The errors in the lattice algorithm are

$$\Delta g_k = \sum_{i=1}^{n+k-2} (b_{k-1,i-1}\Delta f_{k-1,i} + f_{k-1,i}\Delta b_{k-1,i-1}) + e_{g,k} \tag{5.85}$$

$$\Delta\rho_k = -\frac{\Delta g_k + \rho_k\Delta\alpha_{k-1}}{\alpha_{k-1}} - e_{\rho,k} \tag{5.86}$$

$$\Delta f_{k,i} = \Delta f_{0,i} + \sum_{j=1}^{k}(b_{j-1,i-1}\Delta\rho_j + \rho_j\Delta b_{j-1,i-1} + e_{f,j,i} \tag{5.87}$$

$$\Delta b_{k,i} = \Delta b_{0,i-k} + \sum_{j-1}^{k}(f_{j-1,i-k-j}\Delta\rho_i + \rho_j\Delta f_{j-1,i-k+j} + e_{b,j,i-k+j} \tag{5.88}$$

$$\Delta\alpha_k = \Delta\alpha_{k-1} + \rho_k\Delta g_k + g_k\Delta\rho_k e_{\alpha,k} \tag{5.89}$$

Solving for Δg_k in (5.86), substituting it in (5.89) and simplifing the result similarly as in previous case, we have

$$\Delta\alpha_k = \frac{\alpha_k}{\alpha_0}\Delta\alpha_0 + \sum_{i=0}^{k}\alpha_k(-2\rho_i\Delta\rho_i + \frac{e_{\alpha,i}}{\alpha_i} + \rho_i e_{\rho,i}) \tag{5.90}$$

At this point, we are going to discuss the term Δg_k in these formula. Firstly, let us write the Δg_1 as

$$\Delta g_1 = \sum_{i=1}^{n-1}(b_{0,i-1}\Delta f_{0,i} + f_{0,i}\Delta b_{0,i-1}) + e_{g,k}$$

This term is independent on $\Delta\rho_1$.

For Δg_2, we have

$$\Delta g_2 = \sum_{i=1}^{n}(b_{1,i-1}\Delta f_{1,i} + f_{1,i}\Delta b_{1,i-1}) + e_{g,2}$$

Using (5.87) and (5.88), we have

$$\begin{aligned}\Delta f_{1,i} &= \Delta f_{0,i} + b_{0,i-1}\Delta\rho_1 + \rho_1\Delta b_{0,i-1} + e_{f,1,i}\\ \Delta b_{1,i-1} &= \Delta b_{0,i-2} + f_{0,i-1}\Delta\rho_1 + \rho_1\Delta f_{0,i-1} + e_{b,1,i-1}\end{aligned}$$

Using the lattice algorithm to replace the terms $b_{1,i-1}$ and $f_{1,i}$ in the Δg_k, we have the coefficient of $\Delta\rho_1$ as

$$\sum_{i=1}^{n-1}[(\rho_1 f_{0,i-1} + b_{0,i-2})b_{0,i-1} + (f_{0,i} + \rho_1 b_{0,i-1})f_{0,i-1}]$$

Since $b_{0,i} = f_{0,i} = u_i$, $c_i = \frac{1}{n}\sum_{j=0}^{n-1} u_j u_{j-i}$ and $\rho_1 = -\frac{c_1}{c_0}$, we obtained a zero cofficient of $\Delta\rho_1$.

The coefficients of $\Delta\rho_i$ in Δg_k are zero. This is an advantage of lattice algorithm over both Levinson and Schur algorthms. Keeping this in mind, we are not going to consider the term $\Delta\rho_i$ in the errors.

In a manner similar to the Levinson and Schur algorithms, the error can also be classified by terms containing products of ρ_i. The first order approximations are given as

$$\Delta\rho_{k,0} = -\frac{1}{\alpha_{k-1}} \sum_{j=1}^{n+k-2} \left[b_{k-1,j-1}(\Delta u_j + \sum_{i=1}^{k} e_{f,i,j}) + f_{k-1,j}(\Delta u_j + \sum_{i=1}^{k} e_{b,i,j-k+i}) \right] + e_{\rho,k} \quad (5.91)$$

If we need the second order approximation, that is, including the term ρ_i, we have

$$\begin{aligned} \Delta\rho_{k,1} &= \Delta\rho_{k,0} - \frac{\rho_k}{\alpha_0}\Delta\alpha_0 - \rho_k \sum_{i=1}^{k-1} \frac{e_{\alpha,i}}{\alpha_i} \\ &\quad \frac{1}{\alpha_{k-1}} \sum_{j=1}^{n+k-2} \left[b_{k-1,j-1} \sum_{i=1}^{k-1} \rho_i \Delta b_{0,j-i} + f_{k-1,j} \sum_{i=1}^{k-1} \rho_i \Delta f_{0,j-k+i} \right] \end{aligned} \quad (5.92)$$

From this result, one can not get the result given by [12]. The reason is that we removed the term Δg_k in (5.89) by (5.86). If we removed the term $\Delta\rho_k$ in (5.89) by (5.86), we will get

$$\Delta\alpha_k = (1 + \rho_k^2)\Delta\alpha_{k-1} - 2\rho_k \Delta g_k - \rho_k \alpha_{k-1} e_{\rho,k} + e_{\alpha,k}$$

Ignoring all errors except of $\Delta\alpha_{k-1}$, we have

$$\Delta\alpha_k = \Delta\alpha_1 \prod_{j=2}^{k-1} (1 + \rho_j^2).$$

Substituting this into $\Delta\rho_k$ of (5.86), we have

$$\Delta\rho_k = -\frac{\rho_k}{\alpha_{k-1}} \Delta\alpha_1 \prod_{j=0}^{k-1} (1 + \rho_j^2)$$

This is the result given by [12].

Comparing the results, one can conclude that our results are more accurate than the one given in [12]. Another advantage of our method is that our method can be used to get more accurate results by including the terms like $\rho_i \rho_{i_1}$, and so on.

5.4 Finite Analysis of Split Algorithm

5.4.1 Finite Analysis of Split Levinson Algorithm

The errors analysis for split Levinson algorithm is almost same as those for Levinson algorithm. The split Levinson algorithm after quantization becomes

$$\tilde{\tau}_k = \sum_{i=0}^{k} \tilde{c}_i \tilde{s}_{k,i} + e_{\tau,k} \tag{5.93}$$

$$\tilde{\beta}_k = \frac{\tilde{\tau}_k}{\tilde{\tau}_{k-1}} \tag{5.94}$$

$$\tilde{s}_{k+1,i} = \tilde{s}_{k,i} + \tilde{s}_{k,i-1} - \tilde{\beta}_k \tilde{s}_{k-1,i-1} \tag{5.95}$$

$$\tilde{\rho}_k = 1 - \frac{\tilde{\beta}_k}{1 + \tilde{\rho}_{k-1}} \tag{5.96}$$

The errors after quantization can be written as

$$\Delta\tau_k = \sum_{i=0}^{k} [c_i \Delta s_{k,i} + s_{k,i} \Delta c_i] + e_{\tau,k} \tag{5.97}$$

$$\Delta\beta_k = \frac{\Delta\tau_k - \beta_k \Delta\tau_{k-1}}{\tau_{k-1}} + e_{\beta,k} \tag{5.98}$$

$$\Delta s_{k,i} = \Delta s_{k-1,i} + \Delta s_{k-1,i-1} - \beta_{k-1} \Delta s_{k-2,i-1} + s_{k-2,i-1} \Delta\beta_{k-1} + e_{s,k,i} \tag{5.99}$$

If we repeat the formula $\Delta s_{k,i}$ to $\Delta s_{k-1,i}$, $\Delta s_{k-1,i-1}$, $\cdots$, by induction, we can get

$$\Delta s_{k,i} = \sum_{j=1}^{k-1} \sum_{m=1}^{k-j} C_{k-j-1}^{m-1} \left(-\beta_j \Delta s_{j-1,i-m} - s_{j-1,i-m} \Delta\beta_j + e_{s,j+1,i-m+1}\right) \tag{5.100}$$

5.4.2 Finite Analysis of Split Schur Algorithm

The Schur algorithm after quantization has the form

$$\tilde{\beta}_k = \frac{\tilde{v}_{k,0}}{\tilde{v}_{k-1,0}} \tag{5.101}$$

$$\tilde{\rho}_k = 1 - \frac{\tilde{\beta}_k}{1 + \tilde{\rho}_{k-1}} \tag{5.102}$$

$$\tilde{v}_{k+1,j} = \tilde{v}_{k,j} + \tilde{v}_{k,j+1} - \tilde{\beta}_k \tilde{v}_{k-1,j+1} \tag{5.103}$$

The errors of the algorithm after quantization are

$$\Delta\beta = \frac{\Delta v_{k,0} - \beta_k \Delta v_{k-1,0}}{v_{k-1,0}} + e_{\beta,k} \tag{5.104}$$

$$\Delta\rho_k = (1 - \rho_k)\frac{\Delta\rho_{k-1}}{1 + \rho_{k-1}} - \frac{\Delta\beta_k}{1 + \rho_{k-1}} + e_{\rho,k} \tag{5.105}$$

$$\begin{aligned}\Delta v_{k,i} = &\sum_{j=1}^{k-1}\sum_{m=1}^{k-j} C_{k-j-1}^{m-1}\left(-\beta_j \Delta v_{j-1,i+m} - v_{j-1,i+m}\Delta\beta_j + e_{v,j+1,i+m-1}\right) \\ &+ \sum_{m=1}^{k-1} C_{k-2}^{m-1}\left(\Delta v_{1,i+m-1} + \Delta v_{1,i+m}\right)\end{aligned} \tag{5.106}$$

5.4.3 Finite Analysis of Split Lattice Algorithm

The lattice algorithm after quantization becomes

$$2\tilde{\tau}_k = \sum_{i=0}^{n+k-1} \tilde{w}_{k,i}^2 + e_{\tau,k} \tag{5.107}$$

$$\tilde{\beta}_k = \frac{\tilde{\tau}_k}{\tilde{\tau}_{k-1}} + e_{\beta,k} \tag{5.108}$$

$$\tilde{\rho}_k = 1 - \frac{\tilde{\beta}_k}{1 + \tilde{\rho}_{k-1}} + e_{\rho,k} \tag{5.109}$$

$$\tilde{w}_{k+1,i} = \tilde{w}_{k,i} + \tilde{w}_{k,i-1} - \tilde{\beta}_k \tilde{w}_{k-1,i-1} + e_{w,k+1,i} \tag{5.110}$$

The errors of lattice algorithm are

$$\Delta\tau_k = \sum_{i=0}^{n+k-1} w_{k,i}\Delta w_{k,i} + e_{\tau,k} \tag{5.111}$$

$$\Delta\beta_k = \frac{\Delta\tau_k - \beta_k \Delta\tau_{k-1}}{\tau_{k-1}} + e_{\beta,k} \tag{5.112}$$

$$\Delta\rho_k = (1 - \rho_k)\frac{\Delta\rho_k}{1 + \rho_{k-1}} - \frac{\Delta\beta_k}{1 + \rho_{k+1}} + e_{\rho,k} \tag{5.113}$$

$$\Delta w_{k,i} = \sum_{j=1}^{k-1}\sum_{m=1}^{k-j} C_{k-j-1}^{m-1}\left(-\beta_j \Delta w_{j-1,i-m} - w_{j-1,i-m}\Delta\beta_j + e_{w,j+1,i-m+1}\right) + \sum_{m=1}^{k-1} C_{k-}^{m-1}\left(\Delta w_{i-m+1} + \Delta w_{i-m}\right) \quad (5.114)$$

Taking the same steps as lattice algorithm, we can prove that Δr_k is independent on Δd_j, $1 \leq j \geq k-1$. So we simplify the $\Delta w_{k,i}$ into

$$\Delta w_{k,i} = \sum_{j=1}^{k-1}\sum_{m=1}^{k-j} C_{k-j-1}^{m-1}\left(-\beta_j \Delta w_{j-1,i-m} + e_{w,j+1,i-m+1}\right) + \sum_{m=1}^{k-1} C_{k-2}^{m-1}\left(\Delta w_{1,i-m+1} + \Delta w_{1,i-m}\right) \quad (5.115)$$

Chapter 6

On Reduced Polynomial Based Split Algorithms

One of the fundamental problems in linear prediction theory is to compute the predictor polynomial and the reflection coefficients (RCs) for a given real, symmetric, positive-definite Toeplitz matrix (henceforth, referred to simply as Toeplitz matrix). For such a task, in the context of order-recursive algorithms, the Levinson-Durbin algorithm [1, 28] was the most computationally efficient algorithm until recently. Recently, an algorithm termed ' the split Levinson algorithm ' has been described in [8], which requires approximately half the number of MULT and same number of ADD as the Levinson-Durbin algorithm. The split Levinson algorithm is based on the correspondence between the set of polynomials orthogonal on the unit circle and the set of polynomials orthogonal on the interval [-1, 1) of the real line. It is also closely related to the algorithms for zero location with respect to th unit circle described in [27].

The Schur, lattice, and the normalized lattice algorithms may also be employed for the computation of the RCs for a given Toeplitz matrix [29, 21, 30]. Each of these algorithms possesses its own set of desirable features and is closely related to the Levinson-Durbin algorithm. In view of such a close relationship and the existence of the split Levinson algorithm, the split Schur, lattice, and the normalized lattice algorithms have also been described in a recent paper [9], which require significantly less MULT than the Schur, lattice and the normalized lattice algorithms respectively, while the number of ADD remain approximately the same.

In this chapter, we derive another form of computationlly efficient three-term recurrence relation for the computation of RCs (and the predictor polynomial for the case of Levinson algorithm) associated with a given Toeplitz matrix. Following the terminology introduced in [8], we term the algorithms as the split algorithms. These new split algorithms are similar to the split algorithms of [8, 9], even though the computations involved are entirely different. The relationships between the various algorithms are also established in the paper.

It is worthwhile to mention here that the three-term recurrences termed the split Levinson algorithm, derived in this paper and in [8] were perhaps derived in [40] for the first time in the context of the one-dimensional inverse problem of reflection seismology. However, their application to computing the reflection coefficients and the Levinson polynomial in a computationally efficient manner was demonstrated in [8] and is being done here as well. Also, the present work and [8] extend these computational savings to the Schur, lattice, and normalized lattice algorithms. Finally, in this paper, we also describe a new lattice realization of a digital filter and a new computationally efficient algorithm for solving the

Toeplitz linear system $T\underline{x} = \underline{b}$.

This Chapter is organized as follows. Section 6.1 sets up the notation for the sequel and develops the necessary mathematical background for the work. In Section 6.2, we introduce the new split Levinson algorithm and present computationally efficient algorithms for the computation of the RCs and the predictor polynomial. The various interrelationships between the new split Levinson algorithm and the previous Levinson-Durbin and split algorithms are also established. In Section 6.3, the corresponding split Schur algorithm is derived, and in Section 6.4, we derive the corresponding lattice and the normalized lattice algorithms. These algorithms lead to a new lattice structure for finite-impulse-response(FIR) filtering of discrete data. Based on the new split Levinson algorithm, a fast algorithm for solving the general linear system $T\mathbf{x} = \mathbf{b}$ is briefly outlined in Section 6.5. It is worthwhile to mention here that in order to avoid introducing new mathematical terminology, a simple matrix-vector approach has been adopted to describe the various algorithms.

6.1 Notation and Classical Algorithms

The mathematical preliminaries fundamental to the content of this work are presented in this section. The various notations are defined, and certain well known results on Levinson, Schur, lattice, and the normalized lattice algorithms and their connection to Toeplitz matrices are described [1, 28, 29, 21, 30].

6.1.1 Notation

Let $\mathcal{R}$ denote the field of real numbers and T be a Toeplitz matrix of order (n+1), specified in term of its elements in the first column, i.e., $c_i \epsilon \mathcal{R}, i = 0, 1, \cdots, n$. For a given matrix T, let T_k denote the principal submatrix of T of order (k+1). Note that $T_n = T$. A (k+1)-dimensional cloumn vector $\mathbf{x}_k = (x_{k,0} x_{k,1} \cdots x_{k,k})^t$ can equivalently be expressed as a polynomial $x_k(z) = \sum_{i=0}^{k} x_{k,i} z^i$ of degree k and vice versa. The equivalence between the vector notation $\mathbf{x}_k$ and polynomial notation $x_k(z)$ is used throughout this work in order to simplify the form of the expressions. In context of filtering, z^{-1} may be considered as the unit delay. Let J denote the exchange matrix having ones on the antidiagonal and zeros elsewhere. The order of the matrix J is understood to be conformable with the context in which it appears. For a given polynomial, $x_k(z) = \sum_{i=0}^{k} x_{k,i} z^i$, let $\hat{x}_k(z)$ denote the reciprocal polynomial, $\hat{x}_k(z) = \sum_{i=0}^{k} x_{k,k-i} z^i$. In vector notation, we have $\hat{\mathbf{x}}_k = J\mathbf{x}_k$. A vector $\mathbf{x}_k$ (polynomial $x_k(z)$) is called symmetric if $\mathbf{x}_k = \hat{\mathbf{x}}_k$ ($x_k(z) = \hat{x}_k(z)$) and anti-symmetric if $\mathbf{x}_k = -\hat{\mathbf{x}}_k$ ($x_k(z) = -\hat{x}_k(z)$). In either of the two cases, $\mathbf{x}_k$ is completely determined by approximately half the elements ($\frac{k}{2}$ if k is even and ($\frac{k+1}{2}$) if k is odd). For a given vector $\mathbf{x}_k = (x_{k,o} \cdots x_{k,k})^t$,vector $\mathbf{x}_m = (x_{k,i} \cdots x_{k,i+m})^t$ is called a subvector of $\mathbf{x}_k$. Of course, $i \geq 0$, and $i + m \leq k$. For $i = 0$, $\mathbf{x}_m$ is the principal subvector of $\mathbf{x}_k$, and for $i = (k - m)/2$, $\mathbf{x}_m$ is center-subvector of $\mathbf{x}_k$ (k-m is even). Any vector $\mathbf{x}_k$ can be written as sum of a SYM vector $\mathbf{s}_k$ and an ASYM vector $\mathbf{a}_k$, where $\mathbf{s}_k = \frac{1}{2}[\mathbf{x}_k + \hat{\mathbf{x}}_k]$ and $\mathbf{a}_k = \frac{1}{2}[\mathbf{x}_k - \hat{\mathbf{x}}_k] = \mathbf{x}_k - \mathbf{s}_k$. If $\mathbf{x}_m$ is the center-subvector of $\mathbf{x}_k$, then $\hat{\mathbf{x}}_m$ is the center-subvector of $\hat{\mathbf{x}}_k$. From this discussion, it is clear that center-subvector of $\mathbf{x}_k$, i.e., $\mathbf{x}_m$,

can be written as sum of a SYM vector $\mathbf{s}_m$ and an ASYM vector $\mathbf{a}_m$, where $\mathbf{s}_m$ and $\mathbf{a}_m$ are the center-subvectors of $\mathbf{s}_k$ and $\mathbf{a}_k$ respectively. We use $\sum_{\mathbf{x}}^k$ to denote the sum of the elements of $\mathbf{x}_k$, i.e., $\sum_{\mathbf{x}}^k = \sum_{i=0}^k x_{k,i}$, and $\|\mathbf{x}_k\|$ is used to denote Euclidean norm of $\mathbf{x}_k$, i.e., $\|\mathbf{x}_k\| = (\sum_{i=0}^k x_{k,i}^2)^{\frac{1}{2}}$. The Toeplitz matrix T_k satisfies the properties 1) $T_k = T_k^t$, 2) $JT_kJ = T_k$.

6.1.2 The Levinson-Durbin Algorithm

The predictor polynomials $\phi_k(z)$ satisfy the two-term recurrence relation,

$$\phi_k(z) = \phi_{k-1}(z) + \rho_k z\hat{\phi}_{k-1}(z) \tag{6.1}$$

where

$$\rho_k = -\sigma_{k-1}^{-1} \sum_{i=0}^{k-1} c_{k-i}\phi_{k-1,i} \tag{6.2}$$

and

$$\sigma_k = \sigma_{k-1}(1-\rho_k^2) \qquad k = 1, 2, \cdots, n \tag{6.3}$$

Here, $\sigma_k = \frac{detT_k}{detT_{k-1}}$, $\sigma_{-1} = 1$, and $\phi_0(z) = 1$. The scalars $\rho_1, \rho_2, \cdots, \rho_n$ are called the RCs. It is clear from (6.1) that $\phi_{k,0} = \phi_{k-1,0} = \cdots = \phi_{0,0} = 1$, and $\phi_{k,k} = \rho_k$. The corresponding vector ϕ_k satisfies the system:

$$T_k\phi_k = [\sigma_k\ 0\ \cdots\ 0\ ]^t \tag{6.4}$$

and, from (6.2), ϕ_{k-1} satisfies the system:

$$T_k \begin{bmatrix} \phi_{k-1} \\ \\ 0 \end{bmatrix} = [\sigma_{k-1}\ 0\ \cdots\ 0\ -\rho_k\sigma_{k-1}]^t \tag{6.5}$$

Also, substituting z=1 in (6.1), we get,

$$\phi_k(1) = \phi_{k-1}(1) + \rho_k \hat{\phi}_{k-1}(1)$$

Since $\phi_{k-1}(1) = \hat{\phi}_{k-1}(1)$, we have, $\phi_k(1) = (1 + \rho_k)\phi_{k-1}(1)$, or

$$\phi_k(1) = \prod_{i=1}^{k}(1 + \rho_i) \tag{6.6}$$

The above expression was first obtained in [6]. A straightforward analysis shows that this algorithm requires $n^2 + O(n)$ MULT and $n^2 + O(n)$ ADD.

6.1.3 The Schur Algorithm

In essence, the Schur algorithm performs the triangular decomposition of the Toeplitz matrix. The Schur polynomials (SPs) satisfy the two-term recurrence relations,

$$z\theta_{n-k-1}(z) = \theta_{n-k}(z) + \rho_k \Delta_{n-k}(z) \tag{6.7}$$

$$\Delta_{n-k-1}(z) = \rho_k \theta_{n-k}(z) + \Delta_{n-k}(z) \tag{6.8}$$

where

$$\rho_k = -\frac{\theta_{n-k,0}}{\Delta_{n-k,0}} \quad k = 1, 2, \cdots, n \tag{6.9}$$

The initial SPs $\theta_{n-1}(z)$ and $\Delta_{n-1}(z)$ are specified as $\theta_{n-1}(z) = \sum_{i=0}^{n-1} c_{i+1} z^i$, and $\Delta_{n-1}(z) = \sum_{i=0}^{n-1} c_i z^i$. The connection between the Levinson-Durbin and the Schur algorithm is as follows. For the linear transformations,

$$\mathbf{u}_n^{(k)} = T(\phi_{k-1}^t \, 0 \, \cdots \, 0)^t \tag{6.10}$$

and

$$\mathbf{v}_n^{(k)} = T(0\ \hat{\phi}_{k-1}^t\ 0\ \cdots\ 0)^t \tag{6.11}$$

the SPs $\theta_{n-k}(z)$ and $\Delta_{n-k}(z)$ are defined as $\theta_{n-k}(z) = \sum_{i=0}^{n-k} u_{n,k+i}^{(k)} z^i$, and $\Delta_{n-k}(z) = \sum_{i=0}^{n-k} v_{n,k+i}^{(k)} z^i$. Note that the superscript (k) has been used here to demonstrate the dependence of the various quantities on the index k. Using (6.1), (6.4) and (6.5), we obtain the recurrence relations in (6.7)– (6.9). Using (6.4) and (6.5), it is easy to see that we also have the identities,

$$u_{n,0}^{(k)} = v_{n,k}^{(k)}(\Delta_{n-k,0}) = \sigma_{k-1}$$

$$u_{n,k}^{(k)}(\theta_{n-k,0}) = v_{n,0}^{(k)} = -\rho_k \sigma_{k-1}$$

and

$$u_{n,i}^{(k)} = v_{n,i}^{(k)} = 0, \quad i = 1, 2, \cdots, k-1$$

The complexity of the Schur algorithm is same as that of the Levinson-Durbin algorithm.

6.1.4 The Lattice Algorithm

In certain signal processing applications, the Toeplitz matrix appears as the product $T = AA^t$, where A is an $(n+1) \times (N+n)$ order upper triangular Toeplitz matrix of rank n+1, given by,

$$A = \begin{bmatrix} a_0 & a_1 & \cdots & a_n & \cdots & a_{N-1} & 0 & \cdots & 0 \\ 0 & a_0 & \cdots & a_{n-1} & \cdots & a_{N-2} & a_{N-1} & \cdots & 0 \\ \vdots & & & & & & & & \\ 0 & 0 & \cdots & a_0 & \cdots & a_{N-n-1} & a_{N-n} & \cdots & a_{N-1} \end{bmatrix} \tag{6.12}$$

Here $N > n$. Let A_k be the $(k+1) \times (N+k)$ order principal submatrix of A, then we have $T_k = A_k A_k^t$, $k = 0, \cdots, n$. The lattice polynomials (LAPs) satisfy the two-term recurrence relation,

$$e_{N+k-1}(z) = e_{N+k-2}(z) + z\rho_k f_{N+k-2}(z) \tag{6.13}$$

and

$$f_{N+k-1}(z) = \rho_k e_{N+k-2}(z) + z f_{N+k-2}(z) \tag{6.14}$$

where

$$\rho_k = -\sigma_{k-1}^{-1}(\mathbf{f}_{N+k-2}^t 0) \begin{pmatrix} 0 \\ \\ \mathbf{e}_{N+k-2} \end{pmatrix} \qquad k = 1, 2, \cdots, n \tag{6.15}$$

In addition, we also have

$$\sigma_{k-1} = \|\mathbf{f}_{N+k-2}\| = \|\mathbf{e}_{N+k-2}\| \tag{6.16}$$

The initial LAPs are specified as $e_{N-1}(z) = f_{N-1}(z) = \sum_{i=0}^{N-1} a_i z^i$. The connection between the Levinson-Durbin and the lattice algorithm is as follows. For the linear transformation,

$$\mathbf{u}_{N+k-1} = A_k^t (\phi_{k-1}^t \; 0)^t \tag{6.17}$$

and

$$\mathbf{v}_{N+k-1} = A_k^t (0 \; \hat{\phi}_{k-1}^t)^t \tag{6.18}$$

the LAPs $e_{N+k-2}(z)$ and $f_{N+k-2}(z)$ are defined as

$$e_{N+k-2}(z) = \sum_{i=0}^{N+k-2} u_{N+k-1,i} z^i,$$

and

$$f_{N+k-2}(z) = \sum_{i=0}^{N+k-2} v_{N+k-1,i+1} z^i.$$

Using (6.1), (6.4) and (6.5), we obtain the relations in (6.13)–(6.16). Also, premultiplying both sides of (6.17) by A_k , we get,

$$A_k \mathbf{u}_{N+k-1} = T_k(\phi_{k-1}^t \; 0)^t = [\sigma_{k-1} \; 0 \; \cdots \; 0 \; -\rho_k \sigma_{k-1}]^t \tag{6.19}$$

As a result, the computation of ρ_k in (6.15) may alternatively be performed as,

$$\rho_k = -\sigma_{k-1}^{-1} \sum_{i=0}^{N-2} a_i e_{N+k-2,i+k} \tag{6.20}$$

However, the expression in (6.15) is used more frequently. Computation of RCs using the lattice algorithm in (6.13)–(6.15), requires $3Nn + 1.5n^2 + O(N+n)$ MULT and ADD. The recurrence relations may also be used to obtain the lattice realization of the FIR filter having transfer function $\phi(z^{-1})$. Such a realization is shown in Fig.1 for order n=5.

6.1.5 The Normalized Lattice Algorithm

The normalized lattice algorithm is the normalized version of (6.13)–(6.15), where all the vectors have unit norm. Using (6.16), the normalized lattice algorithm may be written as

$$g_{N+k-1}(z) = (1-\rho_k^2)^{-\frac{1}{2}}[g_{N+k-2}(z) + z\rho_k h_{N+k-2}(z)] \tag{6.21}$$

$$h_{N+k-1}(z) = (1-\rho_k^2)^{-\frac{1}{2}}[\rho_k g_{N+k-2}(z) + z h_{N+k-2}(z)] \tag{6.22}$$

where

$$\rho_k = (\mathbf{g}_{N+k-2}^t \; 0) \begin{pmatrix} \cdot & 0 \\ & \\ & \mathbf{h}_{N+k-2} \end{pmatrix} \tag{6.23}$$

Here, $\mathbf{g}_{N+k-2} = \frac{\mathbf{e}_{N+k-2}}{\|\mathbf{e}_{N+k-2}\|}$ and $\mathbf{h}_{N+k-2} = \frac{\mathbf{f}_{N+k-2}}{\|\mathbf{f}_{N+k-2}\|}$, and the initialization is given by $g_{N-1}(z) = h_{N-1}(z) = (\sum_{i=0}^{N-1} a_i^2)^{-\frac{1}{2}} \sum_{i=0}^{N-1} a_i z^i$. The normalized lattice algorithm requires $5Nn + 2.5n^2 + O(N+n)$ MULT, $3Nn + 1.5n^2 + O(N+n)$ ADD, and n square-root computations.

We conclude this section by stating that the computationally efficient alternatives to the above algorithms, which are presented in the following sections, follow a very similar approach for the various computation. The main difference, perhaps, is in the use of a three-term recurrence in place of a two-term recurrence.

6.2 A New Split Levinson Algorithm

In this section, we present a new split Levinson algorithm based on the recursive evaluation of a symmetric polynomial using a three-term recurrence relation. Consider the system of linear equations:

$$T_k \mathbf{q}_k = [\theta_k \ \cdots \ \theta_k]^t \tag{6.24}$$

Where the scalar θ_k is chosen such that $q_{k,0} = 1$. Clearly, $\mathbf{q}_k$ is *symmetric*, and , therefore, $q_{k,k} = 1$. The above system, though relatively unknown in signal processing applications, appears frequently in problems of discrete inverse scattering [31, 32]. In the present context, we are interested only in an efficient computation of the RCs and the predictor polynomial. To obtain a recurrence for $q_k(z)$ we proceed as follows. Writing (6.24) in terms of k-1 and k-2, it is straightforward to see that,

$$T_k \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} = [\theta_{k-1} \ \cdots \ \theta_{k-1} \ \gamma_{k-1}]^t \tag{6.25}$$

$$T_k \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} = [\gamma_{k-1} \ \theta_{k-1} \ \cdots \ \theta_{k-1}]^t \tag{6.26}$$

and

$$T_k \begin{bmatrix} 0 \\ \mathbf{q}_{k-2} \\ 0 \end{bmatrix} = [\gamma_{k-2}\ \theta_{k-2}\ \cdots\ \theta_{k-2}\ \gamma_{k-2}]^t \tag{6.27}$$

Based on (6.25)–(6.27), the recurrence for $\mathbf{q}_k$ can be written as,

$$\mathbf{q}_k = \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} - \alpha_k \begin{bmatrix} 0 \\ \mathbf{q}_{k-2} \\ 0 \end{bmatrix} \tag{6.28}$$

Premultiplying both sides of (6.28) by T_k, we get,

$$\begin{bmatrix} \theta_k \\ \theta_k \\ \vdots \\ \theta_k \\ \theta_k \end{bmatrix} = \begin{bmatrix} \theta_{k-1} \\ \theta_{k-1} \\ \vdots \\ \theta_{k-1} \\ \gamma_{k-1} \end{bmatrix} + \begin{bmatrix} \gamma_{k-1} \\ \theta_{k-1} \\ \vdots \\ \theta_{k-1} \\ \theta_{k-1} \end{bmatrix} - \alpha_{k-1} \begin{bmatrix} \gamma_{k-2} \\ \theta_{k-2} \\ \vdots \\ \theta_{k-2} \\ \gamma_{k-2} \end{bmatrix} \tag{6.29}$$

The above equation is satisfied if the scalar α_k is such that

$$\theta_{k-1} + \gamma_{k-1} - \alpha_k \gamma_{k-2} = 2\theta_{k-1} - \alpha_k \theta_{k-2} \tag{6.30}$$

or

$$\alpha_k = \frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-2} - \gamma_{k-2}} \tag{6.31}$$

Once, α_k is known, θ_k is obtained as,

$$\theta_k = 2\theta_{k-1} - \alpha_k \theta_{k-2} \tag{6.32}$$

Also, (6.28) can be written in the polynomial form as,

$$q_k(z) = (1+z)q_{k-1}(z) - \alpha_k z q_{k-2}(z) \tag{6.33}$$

The recursive computation of $q_k(z)$ from $q_{k-1}(z)$, $k = 2, \cdots, n$, consists in (i) computing γ_{k-1} using (6.25), (ii) computing α_k using (6.31), (iii) computing $q_k(z)$ using (6.33), and (iv) computing θ_k using (6.32). The initial values are given by $q_0(z) = 1$, $\theta_0 = c_0$, $q_1(z) = 1+z$, $\theta_1 = c_0 + c_1$, and $\gamma_1 = c_2 + c_1$. Using the symmetry property of $q_k(z)$, $q_n(z)$ can be computed recursively in $0.5n^2 + O(n)$ MULT and $n^2 + O(n)$ ADD. In the following,we establish the relationships between the polynomials $q_k(z)$, predictor polynomials $\phi_k(z)$, and the polynomials used in the split Levinson algorithm of [8]. We consider the relationship between $q_k(z)$ and $\phi_k(z)$ first. Combining (6.24) and a scaled version of (6.26) (scalar $= -\lambda_k$), we get,

$$T_k \left\{ \mathbf{q}_k - \lambda_k \begin{bmatrix} 0 \\ \\ \mathbf{q}_{k-1} \end{bmatrix} \right\} = \begin{bmatrix} \theta_k - \lambda_k \gamma_{k-1} \\ \theta_k - \lambda_k \theta_{k-1} \\ \vdots \\ \theta_k - \lambda_k \theta_{k-1} \end{bmatrix} \tag{6.34}$$

Setting

$$\lambda_k = \frac{\theta_k}{\theta_{k-1}} \tag{6.35}$$

(6.34) reduces to,

$$T_k \left\{ \mathbf{q}_k - \lambda_k \begin{bmatrix} 0 \\ \\ \mathbf{q}_{k-1} \end{bmatrix} \right\} = \begin{bmatrix} \frac{(\theta_{k-1}-\gamma_{k-1})\theta_k}{\theta_{k-1}} \\ 0 \\ \vdots \\ 0 \end{bmatrix} \tag{6.36}$$

A comparison of (6.4) and (6.36) reveals that,

$$\phi_k(z) = q_k(z) - \lambda_k z q_{k-1}(z) \tag{6.37}$$

and

$$\sigma_k = \frac{(\theta_{k-1} - \gamma_{k-1})\theta_k}{\theta_{k-1}} \tag{6.38}$$

Since $\phi_{k,0} = 1$ and the first element of $\mathbf{q}_k$ in (6.36) is also 1. Also, $\phi_{k,k} = \rho_k$ which, in turn, implies that (refer to the r.h.s. of (6.36) again),

$$\phi_{k,k} = \rho_k = 1 - \lambda_k \tag{6.39}$$

or,

$$\lambda_k = 1 - \rho_k \tag{6.40}$$

Substituting (6.35) in (6.38), we get,

$$\theta_{k-1} - \gamma_{k-1} = \frac{\sigma_k}{\lambda_k} \tag{6.41}$$

Combining (6.35) and (6.40), it can be stated that,

$$\frac{\theta_k}{\theta_{k-1}} = 1 - \rho_k \tag{6.42}$$

or, in a non-recursive form, we have,

$$\theta_k = c_0 \prod_{i=1}^{k}(1-\rho_i) \tag{6.43}$$

Dividing both sides of (6.32) by θ_{k-1} and simplifying using (6.42), we also get

$$\alpha_k = (1-\rho_{k-1})(1+\rho_k) \tag{6.44}$$

Thus, an order-recursive algorithm for the computation of the predictor polynomial $\phi_n(z)$ and the RCs $\rho_1, \rho_2, \cdots, \rho_n$ consists in computing $q_k(z)$, $k = 2, \cdots, n$ recursively as outlined above, computing ρ_k, $k = 1, 2, \cdots, n$ using (6.39), and then recovering $\phi_n(z)$ using $q_n(z)$ and $q_{n-1}(z)$ as in (6.37). This new algorithm is summarized in Table I. The recovery of $\phi_n(z)$ requires an additional 0.5n MULT and n ADD. Of course, if one is interested in computing the RCs only, then (6.37) may not be used. Also, if it is required to compute all the predictor polynomials $\phi_k(z)$, $k = 1, 2, \cdots, n$, then the procedure descibed here may still be used. However, such a procedure would then require $0.75n^2 + O(n)$ MULT and $1.5n^2 + O(n)$ ADD.

Another important relation is obtained by setting z=1 in (6.37) and considering the polynomials $q'_k(z) = \theta_k^{-1} q_k(z)$. This leads to

$$\phi_k(1) = \theta_k \sum_{\mathbf{q}'}^{k} - \lambda_k \theta_{k-1} \sum_{\mathbf{q}'}^{k-1}$$

Using (6.6), (6.35), and (6.43), we get,

$$\sum_{\mathbf{q}'}^{k} - \sum_{\mathbf{q}'}^{k-1} = c_0^{-1} \prod_{i=1}^{k} \frac{1+\rho_i}{1-\rho_i}$$

or

$$\sum_{\mathbf{q}'}^{k} = c_0^{-1} \left[1 + \sum_{j=1}^{k} \left\{ \prod_{i=1}^{j} \frac{1+\rho_i}{1-\rho_i} \right\} \right] \tag{6.45}$$

as $q_0' = c_0^{-1}$. The above relation has been employed in problems of discrete inverse scattering [31, 32, 33] and in problem of computing likelihood ratios for known signals in stationary correlated Gaussian noise [34]. Note that the vector $\mathbf{q}_k'$ is solution to the system,

$$T_k \mathbf{q}_k' = [1\ 1\ \cdots\ 1]^t \tag{6.46}$$

and

$$\sum_{\mathbf{q}'}^{k} = [1\ 1\ \cdots\ 1] T_k^{-1} [1\ 1\ \cdots\ 1]^t \tag{6.47}$$

Since $q_k'(z) = \theta_k^{-1} q_k(z)$, we also have,

$$\sum_{\mathbf{q}}^{k} = \left[\prod_{i=1}^{k} (1-\rho_i) + \sum_{j=1}^{k} \left\{ \prod_{i=1}^{j} (1+\rho_i) \prod_{l=j+1}^{k} (1-\rho_l) \right\} \right] \tag{6.48}$$

Furthermore, substituting z=1 in (6.33), we get the recursion,

$$\sum_{\mathbf{q}}^{k} = 2 \sum_{\mathbf{q}}^{k-1} - \alpha_k \sum_{\mathbf{q}}^{k-2} \tag{6.49}$$

We now turn to the relationship between $q_k(z)$ and the polynomials used in the split Levinson algorithm of [8]. Adding (6.25) and (6.26) and subtracting a scaled version of (6.27) (taking into account $\lambda_{k-1} = \frac{\theta_{k-1}}{\theta_{k-2}}$, we must set the scaler $= 2\lambda_{k-1}$) we get,

$$T_k \left\{ \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} - 2\lambda_{k-1} \begin{bmatrix} 0 \\ \mathbf{q}_{k-2} \\ 0 \end{bmatrix} \right\} = [\tau_k\ 0\ \cdots\ 0\ \tau_k]^t \tag{6.50}$$

where

$$\tau_k = \theta_{k-1} + \gamma_{k-1} - \frac{2\theta_{k-1}\gamma_{k-2}}{\theta_{k-2}} \tag{6.51}$$

Let,

$$\mathbf{p}_k = \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} - 2\lambda_{k-1} \begin{bmatrix} 0 \\ \mathbf{q}_{k-2} \\ 0 \end{bmatrix} \tag{6.52}$$

In polynomial form,

$$p_k(z) = (1+z)q_{k-1}(z) - 2\lambda_{k-1} z q_{k-2}(z) \tag{6.53}$$

Using (6.30), (6.38), (6.42), and (6.44), it may be established that,

$$\tau_k = \frac{\sigma_k}{1+\rho_k} \tag{6.54}$$

and (6.50) can be written as,

$$T_k \mathbf{p}_k = [\tau_k \; 0 \; \cdots \; 0 \; \tau_k]^t \tag{6.55}$$

It is clear that the vector $\mathbf{p}_k$ is symmetric, and $p_{k,0} = 1$. A comparison of (6.54) and (6.55) derived above with (6.28) of [8] reveals that the polynomials $p_k(z)$ as expressed in (6.53) is same as the symmetric polynomial $p_k(z)$ of [8].

Also, subtracting (6.26) from (6.25), we get,

$$T_k \left\{ \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} \right\} = (\theta_{k-1} - \gamma_{k-1})[1 \; 0 \; \cdots \; 0 \; -1]^t \tag{6.56}$$

Let

$$\mathbf{p}_k^* = \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} \tag{6.57}$$

or, in polynomial form,

$$p_k^*(z) = (1-z)q_{k-1}(z) \tag{6.58}$$

and

$$\tau_k^* = \theta_{k-1} - \gamma_{k-1} \tag{6.59}$$

Combining (6.41) and (6.59), we also have

$$\tau_k^* = \frac{\sigma_k}{\lambda_k}$$

Thus, (6.56) may be written as,

$$T_k \mathbf{p}_k^* = [\tau_k^* \; 0 \; \cdots \; 0 \; -\tau_k^*]^t \tag{6.60}$$

It is clear from (6.57) that the vector $\mathbf{p}_k^*$ is antisymmetric, and $p_{k,0}^* = -p_{k,k}^* = 1$. A comparison of (6.58) and (6.60) derived above with (6.21)–(6.23), (6.30) and (6.31) of [8] reveals that the polynomials $q_k(z)$ studied here are *the same* as the reduced polynomials $q_k(z)$ of [8]. The analysis performed here also invalidates some of the statements made in [8] regarding the reduced polynomials.

The numerical stability of the algorithm which processes $q_k(z)$ recursively in place of $\phi_k(z)$ as in the Levinson-Durbin algorithm is an important issue and needs further analysis. Such an issue goes beyond the scope of this paper. Also, it is worthwhile to mention here that the recovery of predictor polynomial $\phi_n(z)$ from the polynomials studied in [8] requires division by (1-z), which is not the case for recovery of $\phi_n(z)$ from $q_n(z)$ and $q_{n-1}(z)$. Therefore, it may be preferable to use the algorithm in terms of $q_k(z)$, $k = 1, 2, \cdots, n$.

Using a similar procedure as described above, we can obtain a three-term recurrence for the polynomial $r_k(z)$ satisfying the system,

$$T_k \mathbf{r}_k = [\theta_k \; -\theta_k \; \cdots \; (-1)^k \theta_k]^t \tag{6.61}$$

Note that, $\hat{\mathbf{r}}_k = (-1)^k \mathbf{r}_k$, i.e., $\hat{\mathbf{r}}_k$ is symmetric if k is even and it is antisymmetric if k is odd. It is interesting to note that the polynomials $r_k(z)$ may be obtained from the polynomials $q_k(z)$ via the change of variables $z \rightarrow -z$. The polynomials $r_k(z)$ satisfy the following three-term recurrence,

$$r_k(z) = (1 - z)r_{k-1}(z) + \alpha_k z r_{k-2}(z)$$

Finally, we conclude this section by stating that the polynomial $q_k(z)$ is a monic polynomial and its recursion may be termed the 'monic recursion'. By an appropriate scaling of $q_k(z)$, one may also obtain the ' balanced recursion ' and the ' dual recursion '[35]. Without any loss of generality, we focus on the monic recursion only.

6.3 The Split Schur Algorithm

In this section, based on the relationships between the classical Levinson-Durbin and Schur algorithms, we derive the split Schur algorithm corresponding to the polynomials $q_k(z)$ for the computation of the RCs.

Consider the linear transformation,

$$\mathbf{u}_n^{(k)} = T(\mathbf{q}_{k-1}^t \; 0 \; \cdots \; 0)^t \tag{6.62}$$

and

$$\mathbf{v}_n^{(k)} = T(0 \; \mathbf{q}_{k-1}^t \; 0 \; \cdots \; 0)^t \tag{6.63}$$

From (6.25) and (6.26), it is clear that

$$u_{n,i}^{(k)} = v_{n,i+1}^{(k)} = \theta_{k-1} \quad i = 0, 1, \cdots, k-1 \tag{6.64}$$

$$u_{n,k}^{(k)} = v_{n,0}^{(k)} = \gamma_{k-1} \tag{6.65}$$

and

$$u_{n,i}^{(k)} = v_{n,i+1}^{(k)} \quad i = k, \cdots, n-1 \tag{6.66}$$

Now, we define the split Schur polynomial as $a_{n-k}(z) = \sum_{i=0}^{n-k} u_{n,k+i}^{(k)} z^i$. As a result $a_{n-k,0} = \gamma_{k-1}$. Using the linear transformation and the various identities in (6.62)–(6.66), the split Levinson recurrence in (6.28) is transformed into the split Schur algorithm. Since the derivation is straightforward, we simply summarize the various expressions in the following:

$$\begin{aligned}
\gamma_{k-1} &= a_{n-k,0} \\
\alpha_k &= \frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-2} - \gamma_{k-2}} \\
z a_{n-k-1}(z) &= (1+z) a_{n-k}(z) - \alpha_k a_{n-k+1}(z) \\
\theta_k &= 2\theta_{k-1} - \alpha_k \theta_{k-2} \\
\rho_k &= 1 - \frac{\theta_k}{\theta_{k-1}} \quad k = 2, \cdots, n
\end{aligned} \tag{6.67}$$

The initialization is given by $a_{n-1}(z) = \sum_{i=0}^{n-1} c_{i+1} z^i$, $a_{n-2}(z) = \sum_{i=0}^{n-2} (c_{i+1} + c_{i+2}) z^i$. The above described split Schur algorithm requires $0.5n^2 + O(n)$ MULT and $n^2 + O(n)$ ADD. It is interesting to note that the relationship between the split version of the Levinson-Durbin and Schur algorithm is same as that of their classical versions.

6.4 The Split Lattice Algorithms

In this section, we derive the split lattice and the split normalized lattice algorithms corresponding to the polynomials $q_k(z)$ for the computation of RCs. Given $T = AA^t$, and

$T_k = A_k A_k^t$, consider the linear transformation,

$$\mathbf{u}_{N+k-1} = A_k^t (\mathbf{q}_{k-1}^t \; 0)^t \tag{6.68}$$

The split lattice polynomials (SLAPs) are defined as, $f_{N+k-2}(z) = \sum_{i=0}^{N+k-2} u_{N+k-1,i} z^i$ or, $\begin{bmatrix} \mathbf{f}_{N+k-2} \\ 0 \end{bmatrix} = \mathbf{u}_{N+k-1}$. From the above transformation, it is clear that,

$$\begin{bmatrix} 0 \\ \mathbf{f}_{N+k-2} \end{bmatrix} = A_k^t \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} \tag{6.69}$$

Also, since $T_k = A_k A_k^t$, premultiplying both sides of (6.68) by A_k, we get,

$$A_k \mathbf{u}_{N+k-1} = T_k (\mathbf{q}_{k-1}^t \; 0)^t = [\theta_{k-1} \; \cdots \; \theta_{k-1} \; \gamma_{k-1}]^t \tag{6.70}$$

or

$$\gamma_{k-1} = \sum_{i=0}^{N-2} a_i f_{N+k-2,i+k} \tag{6.71}$$

One form of the new split lattice algorithm is obtained by using the linear transformations in (6.68) and (6.69) on (6.28). Such a derivation is based on (6.71), and the expressions for the corresponding split lattice algorithm can be summarized as,

$$\begin{aligned} \gamma_{k-1} &= \sum_{i=0}^{N-2} a_i f_{N+k-2,i+k} \\ \alpha_k &= \frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-2} - \gamma_{k-2}} \\ f_{N+k-1}(z) &= (1+z) f_{N+k-2}(z) - \alpha_k z f_{N+k-3}(z) \\ \theta_k &= 2\theta_{k-1} - \alpha_k \theta_{k-2} \\ \rho_k &= 1 - \frac{\theta_k}{\theta_{k-1}} \quad k = 2, \cdots, n \end{aligned} \tag{6.72}$$

The initialization is given by $f_{N-1}(z) = \sum_{i=0}^{N-1} a_i z^i$, $f_N(z) = \sum_{i=0}^{N}(a_i + a_{i-1})z^i$, $\theta_0 = \sum_{i=0}^{N-1} a_i^2$.

Another form of the split lattice algorithm may be obtained by considering the inner product,

$$\begin{aligned}
\mathcal{I}_k &= \left[\mathbf{f}_{N+k-2}^t\ 0\right] \begin{bmatrix} 0 \\ \\ \mathbf{f}_{N+k-2} \end{bmatrix} = \left[\mathbf{q}_{k-1}^t\ 0\right] A_k A_k^t \begin{bmatrix} 0 \\ \\ \mathbf{q}_{k-1} \end{bmatrix} \\
&= \left[\mathbf{q}_{k-1}^t\ 0\right] T_k \begin{bmatrix} 0 \\ \\ \mathbf{q}_{k-1} \end{bmatrix} = \left[\mathbf{q}_{k-1}^t\ 0\right] \left[\gamma_{k-1}\ \theta_{k-1}\ \cdots\ \theta_{k-1}\right]^t
\end{aligned}$$

Since $q_{k-1,0} = 1$, the above expression simplifies to,

$$\mathcal{I}_k = \gamma_{k-1} + \theta_{k-1}(\sum_{\mathbf{q}}^{k-1} -1)$$

or

$$\theta_{k-1} - \gamma_{k-1} = \theta_{k-1} \sum_{\mathbf{q}}^{k-1} - \mathcal{I}_k$$

As a result, the split lattice algorithm based on the above expression may be written as,

$$\begin{aligned}
\mathcal{I}_k &= [\mathbf{f}_{N+k-2}^t\ 0] \begin{bmatrix} 0 \\ \\ \mathbf{f}_{N+k-2} \end{bmatrix} \\
\theta_{k-1} - \gamma_{k-1} &= \theta_{k-1} \sum_{\mathbf{q}}^{k-1} - \mathcal{I}_k \\
\alpha_k &= \frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-2} - \gamma_{k-2}} \\
f_{N+k-1}(z) &= (1+z) f_{N+k-2}(z) - \alpha_k z f_{N+k-3}(z) \qquad (6.73) \\
\theta_k &= 2\theta_{k-1} - \alpha_k \theta_{k-2} \\
\sum_{\mathbf{q}}^{k} &= 2 \sum_{\mathbf{q}}^{k-1} - \alpha_k \sum_{\mathbf{q}}^{k-2} \\
\rho_k &= 1 - \frac{\theta_k}{\theta_{k-1}} \quad k = 2, \cdots, n
\end{aligned}$$

The initializations are $\sum_{\mathbf{q}}^{0} = 1$, $\sum_{\mathbf{q}}^{1} = 2$.

The normalized split lattice algorithm may be based on either (72) or (73). In the following, we focus on (6.73). Consider the norm,

$$\begin{aligned}
\mathcal{N} &= \|\mathbf{f}_{N+k-2}\|^2 = \mathbf{f}_{N+k-2}^t \mathbf{f}_{N+k-2} = [\mathbf{f}_{N+k-2}^t \; 0] \begin{bmatrix} \mathbf{f}_{N+k-2} \\ \\ 0 \end{bmatrix} \\
&= \mathbf{u}_{N+k-1}^t \mathbf{u}_{N+k-1} = [\mathbf{q}_{k-1}^t \; 0] T_k \begin{bmatrix} \mathbf{q}_{k-1} \\ \\ 0 \end{bmatrix} \\
&= [\mathbf{q}_{k-1}^t \; 0][\theta_{k-1} \; \cdots \; \theta_{k-1} \; \gamma_{k-1}]^t = \theta_{k-1} \sum_{\mathbf{q}}^{k-1}
\end{aligned}$$

Let us define,

$$\bar{\mathbf{f}}_{N+k-2} = \frac{\mathbf{f}_{N+k-2}}{\|\mathbf{f}_{N+k-2}\|}$$

Now consider the inner product,

$$\mathcal{I}_{k-1} = [\bar{\mathbf{f}}_{N+k-2} \; 0] \begin{bmatrix} 0 \\ \\ \bar{\mathbf{f}}_{N+k-2} \end{bmatrix} = \frac{\theta_{k-1} \sum_{\mathbf{q}}^{k-1} - (\theta_{k-1} - \gamma_{k-1})}{\theta_{k-1} \sum_{\mathbf{q}}^{k-1}}$$

or

$$\frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-1} \sum_{\mathbf{q}}^{k-1}} = 1 - \mathcal{I}_{k-1}$$

Similarly,

$$\frac{\theta_{k-2} - \gamma_{k-2}}{\theta_{k-2} \sum_{\mathbf{q}}^{k-2}} = 1 - \mathcal{I}_{k-2}$$

Based on the above two relations, we define two scalars a_{k-1} and b_{k-1} as

$$\begin{aligned}
a_{k-1} &= \frac{1 - \mathcal{I}_{k-1}}{1 - \mathcal{I}_{k-2}} = \frac{(\theta_{k-1} - \gamma_{k-1})\theta_{k-2} \sum_{\mathbf{q}}^{k-2}}{(\theta_{k-2} - \gamma_{k-2})\theta_{k-1} \sum_{\mathbf{q}}^{k-1}} \\
b_{k-1} &= \left(\frac{\theta_{k-1} \sum_{\mathbf{q}}^{k-1}}{\theta_{k-2} \sum_{\mathbf{q}}^{k-2}} \right)^{\frac{1}{2}}
\end{aligned}$$

Then $\alpha_k = a_{k-1}b_{k-1}^2$. Replacing each of $\mathbf{f}_i$ by $\|\mathbf{f}_i\|\bar{\mathbf{f}}_i$ in (6.73), we obtain the normalized split lattice algorithm. The various expressions for such an algorithm can be summarized as,

$$\begin{aligned}
\mathcal{I}_{k-1} &= [\bar{\mathbf{f}}_{N+k-2}\ 0]\begin{bmatrix} 0 \\ \bar{\mathbf{f}}_{N+k-2} \end{bmatrix} \\
a_{k-1} &= (1-\mathcal{I}_{k-1})(1-\mathcal{I}_{k-2})^{-1} \\
\alpha_k &= a_{k-1}b_{k-1}^2 \\
\theta_k &= 2\theta_{k-1} - \alpha_{k-2}\theta_{k-2} \\
\sum_{\mathbf{q}}^{k} &= 2\sum_{\mathbf{q}}^{k-1} - \alpha_k \sum_{\mathbf{q}}^{k-2} \\
b_k &= \left[\theta_k \sum_{\mathbf{q}}^{k}(\theta_{k-1}\sum_{\mathbf{q}}^{k-1})^{-1}\right]^{\frac{1}{2}} \\
\bar{f}_{N+k-1}(z) &= b_k^{-1}[(1+z)\bar{f}_{N+k-2}(z) - a_{k-1}b_{k-1}\bar{f}_{N+k-3}(z)] \\
\rho_k &= 1 - \frac{\theta_k}{\theta_{k-1}} \quad k = 2, \cdots, n
\end{aligned} \tag{6.74}$$

The initializations are same as in the split lattice algorithm except that the initial vectors are replaced by their normalized forms. Also, using (6.42), (6.43) and (6.48), it may be established that $b^2 < 6$. Using a similar approach as above, we obtain the nomalized split lattice algorithm based on (72), Such an algorithm consists in the following computations,

$$\begin{aligned}
\gamma_{k-1} &= (\sum_{i=0}^{N-2} a_i \bar{\mathbf{f}}_{N+k-2,i+k})\beta_{k-1} \\
\alpha_k &= \frac{\theta_{k-1} - \gamma_{k-1}}{\theta_{k-2} - \gamma_{k-2}} \\
\theta_k &= 2\theta_{k-1} - \alpha_k\theta_{k-2} \\
\sum_{\mathbf{q}}^{k} &= 2\sum_{\mathbf{q}}^{k-1} - \alpha_k \sum_{\mathbf{q}}^{k-2}
\end{aligned} \tag{6.75}$$

$$
\begin{aligned}
\beta_k &= (\theta_k \sum_{\mathbf{q}}^{k})^{\frac{1}{2}} \\
b_k &= \frac{\beta_k}{\beta_{k-1}} \\
\bar{f}_{N+k-1}(z) &= b_k^{-1}[(1+z)\bar{f}_{N+k-2}(z) - \alpha_k b_{k-1}^{-1}\bar{f}_{N+k-3}(z)] \\
\rho_k &= 1 - \frac{\theta_k}{\theta_{k-1}} \quad k = 2, \cdots, n
\end{aligned}
$$

The split lattice algorithm based on (72) requires $2Nn + 0.5n^2 + O(N+n)$ MULT and $3Nn + n^2 + O(N+n)$ ADD while $2Nn + n^2 + O(N+n)$ MULT and $3Nn + 1.5n^2 + O(N+n)$ ADD are required for the split lattice algorithm based on (6.73). For the normalized split lattice algoirthm described in (6.74), $3Nn + 1.5n^2 + O(N+n)$ MULT, $3Nn + 1.5n^2 + (N+n)$ ADD and n square-roots are required. The normalized split lattice algorithm described in (6.75) requires $3Nn + n^2 + O(N+n)$ MULT, $3Nn + n^2 + O(N+n)$ ADD and n square-roots.

The recurrence relations for the split lattice algorithm described here may be employed to obtain a new lattice realization of the FIR filter having the transfer function $\phi(z^{-1})$. Such a realization is shown in Fig.2 for order n=5. It is important to note that the previously described split algorithms of [8], [9] *cannot* be used to realize the lattice implementation of the FIR filter having the transfer function $\phi(z^{-1})$, since the terminal stage of such a realization will require division by $(1 - z^{-1})$, which corresponds to an unstable IIR filter of order 1 having a pole at $z = 1$..

6.5 Related Algorithms and Extensions

A natural qestion arises at this stage— can similar savings in number of arithmetic operations be obtained for more general forms of Toeplitz matrices, for example, hermitian Toeplitz matrices, non-symmetric Toeplitz matrices, and two-dimensional Toeplitz matrices? Also, another problem closely related to the problem of computing the Levinson polynomial, is to solve for the general linear system $T\mathbf{x} = \mathbf{b}$. In the following, we briefly summarize some of our findings on these questions. A more detailed analysis will form a topic of a future research paper.

For a hermitian Toeplitz matrix T, the recurrence can be developed on vectors satisfying the system,

$$T_k \mathbf{q}_k = [\theta_k \cdots \theta_k]^t \quad k = 1, 2, \cdots, n$$

where θ_k is a *real* scalar. Once again, $\mathbf{q}_k$ satisfies the symmetry property $\mathbf{q}_k = \hat{\mathbf{q}}_k$, where $\hat{\mathbf{q}}_k$ is defined as $\hat{\mathbf{q}}_k = J\mathbf{q}_k^*$. In this case, $q_{k,0} \neq 1$, $k = 1, 2, \cdots, n$. The recusive algorithm based on the computation of $\mathbf{q}_k$, $k = 1, \cdots, n$, provides computational savings similar to the real symmetric case [36].

For a non-symmetric Toeplitz matrix T, a recurrence may be developed on vectors satisfying the system $T_k \mathbf{q}_k = [\theta_k \cdots \theta_k]^t$, where θ_k is a real scalar. However, in this case, the vector $\mathbf{q}_k$ possesses no symmetry properties and, as a result , no computational savings are possible over the Levinson-Durbin algorithm. The derivation of such an algorithm is straightforward and is omitted here. For the two-dimensional case, there are no computational savings as well, unless additional constraints are imposed on the mathematical

structure of the matrix.

The split Levinson algorithm based on recursive computation of symmetric vectors $\mathbf{q}_k$ as described here may also be employed to solve for the linear system $T\mathbf{x} = \mathbf{b}$ in an efficient manner. An outline of the algorithm is as follows.

The data vector $\mathbf{b}$ can written as a sum of a symmetric vector $\mathbf{s}$ and an anti-symmetric vector $\mathbf{a}$, where $\mathbf{s} = \frac{1}{2}[\mathbf{b} + \hat{\mathbf{b}}]$, and $\mathbf{a} = \mathbf{b} - \mathbf{s} = \frac{1}{2}[\mathbf{b} - \hat{\mathbf{b}}]$. If $\mathbf{y}$ is solution to the system $T\mathbf{y} = \mathbf{s}$ and $\mathbf{z}$ is solution to the system $T\mathbf{z} = \mathbf{a}$, then solution to the system $T\mathbf{y} = \mathbf{b}$ can simply be written as,

$$\mathbf{x} = \mathbf{y} + \mathbf{z} \tag{6.76}$$

Thus, the task of solving for $T\mathbf{x} = \mathbf{b}$ is reduced to solving two tasks, that is, $T\mathbf{y} = \mathbf{s}$ and $T\mathbf{z} = \mathbf{a}$ at a cost of 2n ADD and 0.5n MULT. Since $\mathbf{s}$ and $\mathbf{a}$ are symmetric and antisymmetric vectors respectively, the corresponding solution vectors $\mathbf{y}$ and $\mathbf{z}$ are also symmetric and antisymmetric respectively.

Combining (6.24) and a scaler version of (6.27) (scalar $= -\frac{\theta_k}{\theta_{k-2}}$), and letting,

$$\mathbf{p}_k = \mathbf{q}_k - \frac{\theta_k}{\theta_{k-2}} \begin{Bmatrix} 0 \\ \mathbf{q}_{k-2} \\ 0 \end{Bmatrix} \tag{6.77}$$

we get

$$T_k\mathbf{p}_k = [\tau_k 0 \cdots 0 \tau_k]^t \tag{6.78}$$

where

$$\tau_k = \frac{\sigma_k}{(1 + \rho_k)} \tag{6.79}$$

Similarly, recall from (6.56) – (6.60), that

$$T_k \mathbf{p}_k^* = [\tau_k^* 0 \cdots 0 - \tau_k^*] \tag{6.80}$$

where

$$\mathbf{p}_k^* = \begin{bmatrix} \mathbf{q}_{k-1} \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ \mathbf{q}_{k-1} \end{bmatrix} \tag{6.81}$$

and

$$\tau_k^* = \frac{\sigma_k}{(1-\rho_k)} \tag{6.82}$$

Given the vector $\mathbf{q}_k$, computation of the vectors $\mathbf{p}_k$ and $\mathbf{p}_k^*$ using (6.77) and (6.81) require a total of k MULT and k ADD. Note that the vectors $\mathbf{p}_k$ and $\mathbf{p}_k^*$ are symmetric and anti-symmetric respectively.

The vectors $\mathbf{p}_k$ and $\mathbf{p}_k^*$ can be employed directly for solving the systems $T\mathbf{y} = \mathbf{s}$ and $T\mathbf{z} = \mathbf{a}$ respectively. The procedure for solving $T\mathbf{y} = \mathbf{s}$ (or $T\mathbf{z} = \mathbf{a}$) consists in computing the solution corresponding to all the center-subvectors of $\mathbf{s}$ (or $\mathbf{a}$). Since all the center-subvectors of $\mathbf{s}$ (or $\mathbf{a}$) are symmetric (or anti-symmetric), the corresponding solutions are also symmetric (or anti-symmetric). The algorithms for solving $T\mathbf{y} = \mathbf{s}$ using $\mathbf{p}_k$ and for solving $T\mathbf{z} = \mathbf{a}$ using $\mathbf{p}_k^*$ are described in the literature [37], [38], and the reader is referred to them for further details. Once $\mathbf{z}$ and $\mathbf{y}$ are computed, the solution vector $\mathbf{x}$ is obtained using (6.76).

The computational complexity of the algorithm for solving $T\mathbf{x} = \mathbf{b}$ using the approach outlined above is $1.125n^2$+O(n) MULT and $2n^2$+O(n) ADD. This represents a small improvement in computational complexity over the algorithm described in [39] which requires

$1.25n^2 + O(n)$ MULT and $2n^2 + O(n)$ ADD. However, there is another advantage in using the approach described here. We solve for the systems $T\mathbf{y} = \mathbf{s}$ and $T\mathbf{z} = \mathbf{a}$ by recursively solving $T_m\mathbf{y}_m = \mathbf{s}_m$ and $T_m\mathbf{z}_m = \mathbf{a}_m$, where $\mathbf{s}_m$ and $\mathbf{a}_m$ are center-subvectors of $\mathbf{s}$ and $\mathbf{a}$ respectively. Therefore, $\mathbf{y}_m$ and $\mathbf{z}_m$ are solutions to SYM and ASYM part of the center-subvectors, $\mathbf{b}_m$, of the data vector $\mathbf{b}$ respectively. Solution to the subsystem $T_m\mathbf{x}_m = \mathbf{b}_m$ can simply be obtained as $\mathbf{x}_m = \mathbf{y}_m + \mathbf{z}_m$. In contrast, the algorithm in [39] recursively computes solution to the system $T_k\mathbf{x}_k = [b_0^{(k)} b_1 \cdots b_k]^t$, where $b_0^{(k)} \neq b_0$. As a result, the intermediate solutions $\mathbf{x}_1, \cdots, \mathbf{x}_n$ are not solutions to any subvector of data vector $\mathbf{b}$. Therefore, the algorithm in [39] may not be applicable in system applications that require solution to subsystems as well.

Chapter 7

Conclusions

Our goal has been to bring together the richness of results available in the theories of orthogonal polynomials and matrices as they relate to Levinson, Schur, and lattice algorithms and the underlying mathematical structure of the Toeplitz matrices. Emphasis is placed on exploiting the properties of the Toeplitz matrices to derive computationally efficient algorithms for solving the associated linear system and computing other parameters of interest.

In our work, we have studied various numerical algorithms for Toeplitz matrices from the standpoint of (i) computational complexity, and (ii) numerical stability and finite precision arithmetic implementation. In the domain of computational complexity, we have derived fast algorithms for testing the strict and wide sense stability of discrete time systems, computing the reflection coefficients and solving one and two dimensional Toeplitz linear system. A superfast algorithm is also derived for solving a block Toeplitz linear system. A new lattice realization of a digital filter is also obtained. In the domain of numerical

stability, it is shown that the recently reported split Levinson algorithm is weakly stable. The finite precision implementation of the algorithms results in errors being introduced. We have analyzed the Levinson, Schur , and lattice algorithms for the effects of finite precision arithmetic on their implementation.

A number of linear algebra problems related to Toeplitz matrices remain open at this time, one of the most significant being the design of a fast Fourier transform based superfast algorithm for solving a near-to-Toeplitz linear system. Also, the application of the new algorithms to computing the eigendecomposition of the Toeplitz matrices must be examined. The parallel processing aspects of the various algorithms and their suitability for VLSI implememtation also needs to studied for solving large scale problems.

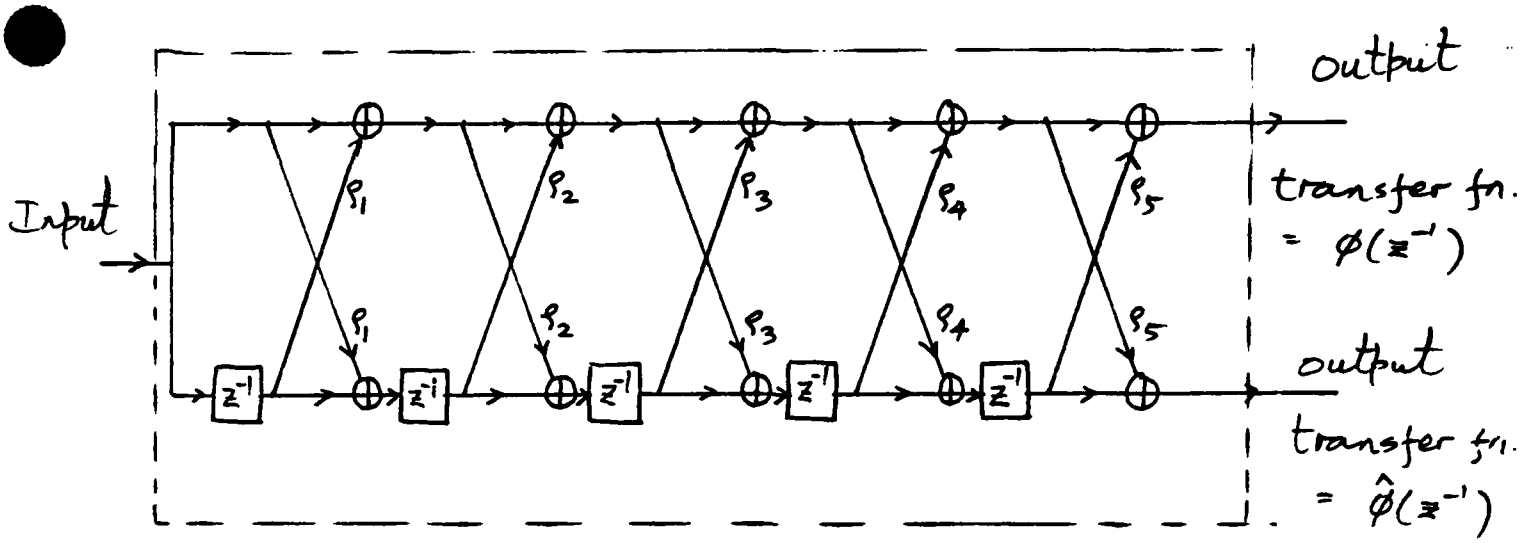


Figure 1. Lattice realization of a FIR filter of order 5; transfer fn. $\phi(z^{-1})$

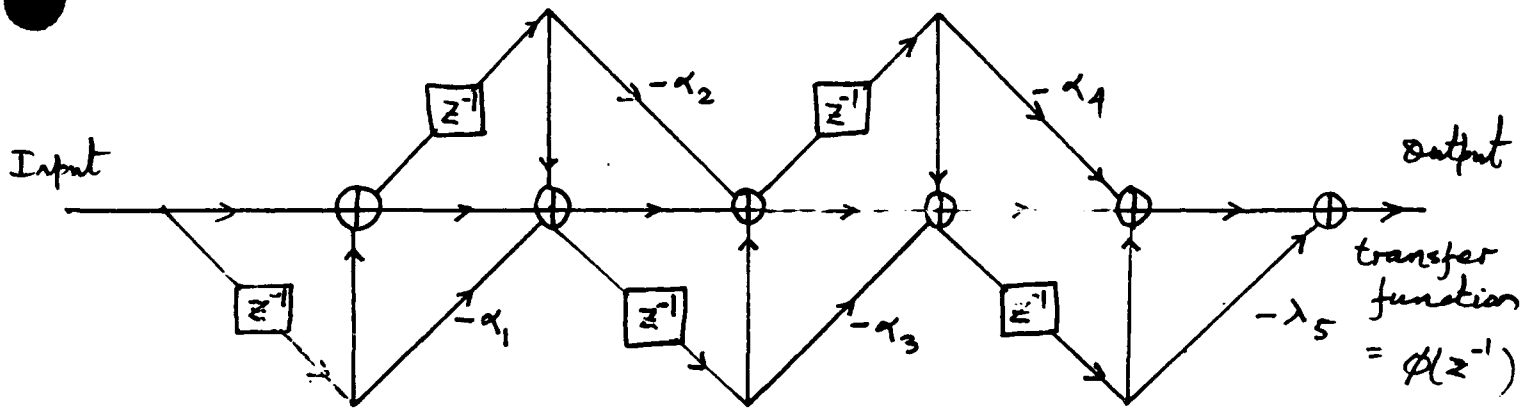


Figure 2. A new lattice realization of a FIR filter of order 5; transfer fn. $\phi(z^{-1})$

Bibliography

[1] N. Levinson, " The Wiener rms (root mean square) error criterion in filter design and prediction ", J. Math. Phys. vol. 25, pp. 261-278, 1946.

[2] T. Kailath, S. Y. Kung and M. Morf, " Displacement ranks of matrices and linear equations ", J. Math and Appl. 68(2):395-407, April, 1979.

[3] M. Benidir and B. Picinbono, " Extensions of the Stability Criterion for ARMA Filters ", IEEE Trans. Acoust. Speech, and Signal Processingn, vol. ASSP-35, No. 4, April 1987.

[4] S. M. Kay and S. L. Marple, Jr, " Spectrum Analysis - A Mordern Perspective ", Proc. IEEE, vol. 69, no. 11, pp. 1380-1419, Nov. 1981.

[5] H. Krishna and Y. Wang, " On Fast Algorithm for Testing the Stability of Discrete Time Systems ", Signal Processing 17 (1989) 251-257.

[6] G. Cybenko, " The Numerical Stability of the Levinson-Durbin Algorithm for Toeplitz Systems of Equations ", SIAM J. Sci. Stat. , Comput. 1:303-319, 1980.

[7] J. R. Bunch, " The Weak and Strong Stability of Algorithms in Numerical Linear Algebra ", Linear Algebra and Its Applications 88/89:49-66, 1987.

[8] P. Delsarte and Y. U. Genin, " The Split Levinson Algorithm ", IEEE Trans. Acoust. , Speech, Signal Processing, vol. ASSP-34, pp. 470-478, June 1986.

[9] P. Delsarte and Y. U. Genin, " On the splitting of classical algorithm in linear prediction theory ", IEEE Trans. Acoust. Speech, Signal Processing, vol. ASSP-35, pp. 645-653, May 1987.

[10] F. de Hoog, " A New Algorithm for Solving Toeplitz Systems of Equations ", Linear Algebra and Its Applications. 88/89:123-138, 1987.

[11] Y. C. Lim, " On the Synthesis of IIR Digital Filters Derived from Single Channel AR Lattice Network ", IEEE Trans. Acoust. Speech, and Signal Processing, vol. ASSP-32, pp. 741-749, Aug. 1984.

[12] S. T. Alexander and Zong M. Rhee, " Analytical Finite Precision Results for Burg's Algorithm and the Autocorrelation Method for Linear Prediction ", IEEE Trans. Acoust. Speech, and Signal Processing, vol. ASSP-35, pp. 626-635, May 1987.

[13] G. E. P. Box and G. M. Jenkins, *Time Series Analysis: forecasting and control.* Holden-Day, San Francisco, 1970.

[14] J. D. Markel and A. H. Gray, "Fixed-point Truncation Arithmetic Implementation of a Linear Prediction Autocorrelation Vocoder," IEEE Trans. Acoust., Speech, and Signal Processing, vol. ASSP-22, No. 4, pp. 273–282, Aug. 1974.

[15] R. A. Horn and C. A. Johnson, *Matrix Analysis.* Cambridge, 1985.

[16] H.Akaike, " Block Toeplitz Matrix Inversion ", SIAM J. Appl. vol. 24, No. 2, March 1973.

[17] I.C.Gohberg and G.Heinig, " Inversion of finite Toeplitz matrices with entries being elements from a noncommutative algebra ", Rev. Roumaine Math. Pures Appl. XIX, No. 5, pp 623–663, 1974.

[18] M. Wax and T. Kailath, " Efficient Inversion of Toeplitz–Block–Toeplitz Matrix ", IEEE Trans. Acoust., Speech, Signal Processing,vol. ASSP–31, No. 5, 1983.

[19] G.A.Merchant and T.W.Parks, " Efficient Solution of a Toeplitz–Plus–Hankel Coefficient Matrix System of Equations ", IEEE Trans. Acoust., Speech,Signal Processing,vol. ASSP–30, No. 1, 1982.

[20] J. Makhoul, "Linear Prediction: A tutorial review," Proc. IEEE, vol. 63, pp. 561–580, 1975.

[21] F. Itakura and S. Saito, " Digital filtering techniques for speech analysis and synthesis, " in Proc. 7th Int. Congr. Acoust., Budapest, 1971, pp. 261–264.

[22] V. Eveleigh, Introduction to Control Systems Design, McGraw Hill, New York, 1987.

[23] E. I. Jury, Theory and Application of the z Transform Method , John Wiley, New York, 1964.

[24] J. M. Travassos-Romano and M. Bellanger, " Zeros and poles of linear prediction digital filters ", in: I. T. Young et al., ed., Signal Processing III. Theories and Applications , (Proc. EUSIPCO–86, 3rd. Eur. Signal Process. Conf., The Hague, The Netherlands, September 1986), Elsevier, Amsterdam, 1986.

[25] H. Krishna and S. D. Morgera, " Computationally efficient stepup and stepdown procedures for real prediction coefficients ", IEEE Trans. Acoust., Speech Signal Process., Vol. 36, No. 8, August 1988, pp. 1353–1355.

[26] H. Krishna and Y. Wang, " Fast Algorithm For Wide Sense Stability of ARMA Filters ", ICASSP-88, pp. 1834-1837.

[27] Y. Bistritz, " Zero location with respect to teh unit circle of discrete–time linear system polynomials ", Proc. IEEE, Vol. 72 , September 1984, pp. 1131–1142.

[28] J.Durbin, " The fitting of time-series models ", *Rev. Inst. Int. Stat.*, vol. 28, pp. 233-244, 1960.

[29] J.LeRoux and C.Gueguen, " A fixed point computation of partial correlation coefficients ", *IEEE Trans. on Acoust., Speech, and Sig. Proc.*, vol. ASSP-25, pp. 257-259, 1977.

[30] A.H.Gray and J.D.Markel, " A normalized digital filter structure ", *IEEE Trans. on Acoust., Speech, and Sig. Proc.*, vol. ASSP-23, no. 3, pp. 268-277, June 1975.

[31] R.E.Caflisch, " An inverse problem for Toeplitz matrices and the synthesis of discrete transmission lines ", *Lin. Alg. and Appl.*, no. 38, pp. 207 -225, 1981.

[32] A.Brukstein and T.Kailath, " Some matrix factorization identities for discrete inverse scattering " , *Lin. Alg. and Appl.*, no. 74, pp. 157-172, 1986.

[33] B.W.Dickinson, " An inverse problem for Toeplitz matrices ", *Lin. Alg. and Appl.*, no. 59, pp. 79-83, 1984.

[34] B.W.Dickinson, " Properties and applications of Gaussian autoregressive processes in detection theory ", *IEEE Trans. INfo. Theory*, vol. IT-27 , no. 3 pp. 343-347, May 1981.

[35] Y.Bistritz, H.Lev-Ari, and T.Kailath, " Immitance-domain Levinson algorithms ", *Int. Conf. Acoust., Speech, and Sig. Proc.*, Tokyo, Japan, April 1986.

[36] B. Krishna,and H. Krishna, " Computationally efficient reduced polynomial based algorithms for hermitian Toeplitz matrices", *SIAM Journal on Applied Mathematics*, accepted for publication.

[37] A.K.Kok, D.G.Manolakis and U.K. Ingle, " Efficient algorithms for 1-D and 2-D non-causal autoregressive system modelling ", *IEEE Int. Conf. on Acoust., Speech, and Sig. Proc.*, Dallas, 1987.

[38] D.C.Forden and J.R.Bellegarda, " A fast algorithm for the recursive design of linear phase filters ", *IEEE Int. Conf. on Acoust., Speech, and Sig. Proc.*, Dallas, 1987.

[39] H.Krishna and S.D.Morgera, " The Levinson recurrence and fast algorithms for solving Toeplitz system of linear equations ", *IEEE Trans. on Acoust., Speech, and Sig. Proc.*, vol. ASSP. 35, No. 6, pp. 839-848, June 1987.

[40] K. P. Bube and R. Burridge, " The one-dimensional inverse problem of reflection seismology, " *SIAM Review*, vol. 25, No. 4, pp. 497–559, Oct. 1983.